NPS ARCHIVE
1969
COOPER, J.

THE GENERAL INFORMATION PROCESSING SYSTEM (GYPSY)

John Duffield Cooper, Jr.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

United States
Naval Postgraduate School

THESIS

THE GENERAL INFORMATION PROCESSING SYSTEM (GIPSY)

by

John Duffield Cooper, Jr.

December 1969

This document has been approved for public release and sale; its distribution is unlimited.

T135574

DUDLEY KNOX L
NAVAL POSTGR
MONTEREY C

The General Information Processing System (GIPSY)

by

John Duffield Cooper, Jr.
Lieutenant Commander, United States Navy
B.S., University of Missouri, 1957

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1969

Thesis C7404 c.1 NPS ARCHIVE 1969 COOPER, J.

ABSTRACT

This paper describes the General Information Processing System, a computer program designed to serve a variety of information processing applications. An input deck to the program is composed of a data base and a description of the processing tasks to be performed on that data. A typical task would be to screen the data base according to given criteria and then output information from the data that met the criteria. For output, the system has flexible formatting capabilities.

Included in the paper, in Section II, is an example of a bibliographical application, complete with a listing of the input deck and the output that was produced. Sections III through VII contain detailed instructions on how to prepare an input deck. A description of system implementation is contained in Section VIII.

IBRARY
AVAL POSTGRADUATE SCHOOL
ONTEREY, CALIF. 93940

TABLE OF CONTENTS

I. INTRODUCTION ---------------------------------- 7
II. A BIBLIOGRAPHY EXAMPLE ------------------------- 10
    A. THE INPUT DATA ---------------------------- 10
    B. PROCESSING THE DATA ------------------------ 14
    C. THE BIBLIOGRAPHY DATA FORM ---------------- 22
III. THE INPUT DECK -------------------------------- 24
    A. OS/360 REQUIREMENTS ------------------------ 24
    B. SETTING UP THE PROGRAM -------------------- 26
    C. NAMING THE ENTRIES ------------------------- 28
    D. INPUT DECK LISTING ------------------------- 29
IV. THE DATA TO BE PROCESSED ---------------------- 30
    A. THE DATA CARDS ----------------------------- 30
    B. THE "IDENT" FIELD -------------------------- 31
V. THE OUTPUT FORMAT ------------------------------ 33
    A. FORM OF THE LINE --------------------------- 34
    B. FORMAT PARAMETER FIELDS ------------------- 35
    C. CONTENT OF THE LINE ------------------------ 37
    D. ORGANIZATION OF THE FORMAT SPECIFICATIONS ----------------------------- 41
VI. SCREENING -------------------------------------- 43
    A. SIMPLE BOOLEAN EXPRESSIONS ----------------- 43
    B. COMPLEX BOOLEAN EXPRESSIONS --------------- 45
    C. THE BOOLEAN EXPRESSION CARDS -------------- 46
VII. OTHER CONTROL CARDS --------------------------- 47
    A. *CHANGE (Change Variable's Value) --------- 47
    B. *ORDER (Reorder Data Records) ------------- 48

C. *PRINT (Print Output) -------------------- 49

D. *EOF (End of File) ------------------------ 50

E. *ENDJOB (End of Job) --------------------- 50

F. *HEADING (Print Heading) ----------------- 51

G. *PRINTCOUNTER (Print "Counter") ---------- 51

H. *TIME (Print Elapsed Time) --------------- 52

VIII. SYSTEM IMPLEMENTATION ------------------------ 53

A. SETTING UP THE PROGRAM ------------------- 53

B. PROCESSING THE DATA FILE ----------------- 56

C. THE FORMAT SPECIFICATIONS ---------------- 58

D. THE SCREENING EXPRESSION ----------------- 63

1. The Screen Table --------------------- 64

2. Expression "Pre-Evaluation" ---------- 67

E. THE REORDERING PROCEDURE ----------------- 69

F. OUTPUT PROCESSING ------------------------ 70

1. Screening the Records ---------------- 72

2. Formatting the Output ---------------- 73

3. Printing the Output ------------------ 74

G. ELAPSED TIME PROCESSING ------------------ 76

IX. CONCLUDING REMARKS --------------------------- 77

COMPUTER PROGRAM ------------------------------------ 78

LIST OF REFERENCES ---------------------------------- 106

INITIAL DISTRIBUTION LIST -------------------------- 107

FORM DD 1473 -- 109

LIST OF FIGURES

1. Bibliography Example Input Deck Listing -------- 11
2. Bibliography Example Data Record 1 ------------- 16
3. Bibliography Example Data Record 2 ------------- 17
4. Bibliography Example Data Record 3 ------------- 18
5. Bibliography Example Bibliography Output ------- 19
6. Bibliography Example Screened Output ----------- 21
7. The Bibliography Data Form --------------------- 23
8. Data File Organization ------------------------- 57
9. The Format Tables ------------------------------ 61
10. Processing a Screen --------------------------- 65

I. INTRODUCTION

Intelligent decision making is predicated on the availability of certain requisite information. It is this "availability of requisite information" that has generated a great deal of interest in an area of computer usage called information retrieval. Information retrieval systems operate on a data base that may be either static or dynamic. However, they usually conform to some prescribed organizational criteria. It is this organization of the data that allows the system to access the desired information.

Most bodies of information (data files) can be organized into logical entities (data records). For each entity (data record) there are a number of attributes (record entries), the values of which describe that entity. For example, a bibliography is a body of information (data file) in which publications (data records) are described by the attributes, author, title, subject, etc., (record entries). For another example, a membership file (data file) is a body of information which describes persons (data records) by name, address, profession, etc., (record entries). Mailing lists, patient files, physical assets inventories, and many other bodies of information could readily be cited here, also.

Within the realm of information retrieval are Management Information Systems, Command and Control Systems, and Airline Reservation Systems, just to mention some of the more significant applications. Due to the magnitude and complexity of these applications special purpose, computer programs are written for them so that they can be made to operate more efficiently. There is, however, a large class of applications which cannot justify the expense of a special purpose system but still require the capability of information retrieval. To meet this need a group of General Information Retrieval Systems have emerged. These systems are designed to be used for a number of different general applications.

References 5, 8, 9, and 10 are fairly detailed descriptons of General Information Retrieval Systems. Reference 7 is an excellent paper devoted to this subject and includes discussion of 19 such systems. Reference 1 describes the BASIC INFORMATION PROCESSOR which was the forerunner of the system presented in this paper.

The General Information Processing System (GIPSY) described here is designed to perform this type of non-arithmetic processing. It was intended to be flexible enough that it might be used for numerous applications and simple enough for user convenience. The system was implemented in PL/1 [Refs. 2, 3, and 4] since its capabilities along these lines simplified the programming chore.

This paper will begin with an illustrative example of GIPSY applied to a bibliography, to give the reader an idea of how it is used and what it can do. Then the details about the input deck, the data to be processed, the output format, screening, and other control cards will be discussed. The manner in which the system was implemented will be described next, followed by a few brief concluding remarks.

II. A BIBLIOGRAPHY EXAMPLE

The application which motivated the development of GIPSY was the maintenance of a bibliography for research. This bibliography contained several hundred references. However, three were chosen as being typical of the references in the bibliography. These three references were used in the development and testing of GIPSY and, therefore, were convenient to use in the following example.

A. THE INPUT DATA

In this example, the data base consists of the entire deck of cards used as data for GIPSY. The data file is made up of the data describing these three references, thus each reference is one record in the data file. Each record is, in turn, described by twelve entries, the twelfth entry is a remarks section and could contain as many as seven paragraphs. Figure 1 contains a listing of the input deck used for this example.

A glance at Figure 1 reveals the general make-up of the input deck. The cards that have a '*' punched in column one are control cards which cause GIPSY to branch to that section of the program which is to process the cards that follow until the next control card is encountered. All cards are begun in column one and any continuation cards are begun in column six. The '/' (slash) character is a reserved, special character, and as such is used by GIPSY

```
*SETUP
     CARDS = 35, ESTIMATE = 5, ENTITY = 19;
AUTHOR/TITLE/PUBLICATION/PUBLISHER/DATE/PAGES/PRICE/REFERNO/
     CITED/LOCATION/DESCRIPTORS/REMARKS/R1/R2/R3/R4/R5/R6/R7
*DATA
FISHMAN, GEORGE S./PROBLEMS IN THE STATISTICAL ANALYSIS OF SIMULATION        F004
     EXPERIMENTS - THE COMPARISON OF MEANS AND THE LENGTH OF SAMPLE          F004
     RECORDS/MEMORANDUM RM-4880-PR/THE RAND CORPORATION, SANTA MONICA,       F004
     CALIF./FEB. 1966/22////GEH, MS/SIM STAT, SPECT ANAL/SHOWS HOW TO        F004
     ESTIMATE THE LENGTHS OF SAMPLE RECORDS NEEDED TO USE CERTAIN            F004
     LARGE SAMPLE RESULTS IN MEASURING STABILITY, DESCRIBES ANALOGIES        F004
     BETWEEN AUTOCORRELATED DATA AND INDEPENDENT OBSERVATIONS, AND           F004
     SUGGESTS A WAY TO TEST THE DIFFERENCE OF THE MEAN OF TWO                F004
     EXPERIMENTS. THIS IS AN APPLICATION OF SPECTRAL ANALYSIS TO             F004
     SIMULATION./   RATHER CLEARLY WRITTEN, BUT COULD USE MORE               F004
     EXPLANATION. THE EXAMPLE INCLUDED DOES NOT REALLY SHOW THE              F004
     POWER OF THE TECHNIQUE.                                                 F004
HOGGATT, AUSTIN C., AND BALDERSTON, FREDERICK E. (EDS.)//SYMPOSIUM ON        H002
     SIMULATION MODELS - METHODS AND APPLICATIONS TO THE BEHAVIORAL          H002
     SCIENCES/SOUTHWESTERN PUBLISHING CO., CINCINNATI, OHIO/1963/289/        H002
     $4.25///ESJ/SIM METH, SIM COG PROC, SIM MOD/A COLLECTION OF 14          H002
     PAPERS PRESENTED AT A CONFERENCE ON COMPUTER SIMULATION HELD AT         H002
     THE UNIVERSITY OF CALIFORNIA, LOS ANGELES, IN SEPTEMBER 1961.           H002
     (SEE C001, H003)                                                        H002
RAPHAEL, BERTRAM/THE STRUCTURE OF PROGRAMMING LANGUAGES/COMM. ACM 9,2//      R003
     FEB. 1966/67-71////GEH, YCC/COMP PROG, LIST PROC/IDENTIFIES THE MAJOR   R003
     COMPONENTS OF PROGRAMMING LANGUAGES, DESCRIBES ALTERNATIVE FORMS        R003
     AVAILABLE FOR IMPLEMENTING THEM, AND COMPARES AND EVALUATES THE         R003
     VARIOUS FORMS. (19 REFERENCES)                                          R003
*EOF
*FORMAT
/                         AUTHOR        /
                                                            /     / 12   1 54   43 100
AUTHOR/
/                         TITLE         /
                                                           /      /  2   1 54   43 100
TITLE/
/                         PUBLICATION   /
                                                           /      /  2   1 54   43 100
PUBLICATION/
/                         PUBLISHER     /
                                                           /      /  2   1 54   43 100
PUBLISHER/
/                         DATE          /                  PAGES  /                PRIC
E  /                                                       /      /  2   1 54   43 100
DATE/PAGES/PRICE/
```

Figure 1

```
/                        REF. NO.      /                                    LOCATED
T   /                                                      /     /  4   1 54   43 100
IDENT/LOCATION/
/                        DESCRIPTORS   /
                                                           /     /  2   1 54   43 100
DESCRIPTORS/
/                        REMARKS       /
                                                                 /  4   1 54   43 100
REMARKS/
/                                      /
                                                           /     /  2   1 54   43 100
R1/
/                                      /
                                                           /     /  2   1 54   43 100
R2/
/                                      /
                                                           /     /  2   1 54   43 100
R3/
/                                      /
                                                           /     /  2   1 54   43 100
R4/
/                                      /
                                                           /     /  2   1 54   43 100
R5/
/                                      /
                                                           /     /  2   1 54   43 100
R6/
/                                      /
                                                           /     /  2   1 54   43 100
R7/                                                             /
/                                                          /       54   1 54   43 100
PAGENO/
*EOF
*PRINT
*CHANGE
          COUNTER = 0;
*FORMAT
/                                                         B I B L I O G R A P H Y
                                                          / / / / 12   1 54   43 100
/                        /     /
                                                           /        3   1 52   31 100
COUNTER./AUTHOR,// "TITLE,"// PUBLICATION,// PUBLISHER,//
    <(/-PUBLISHER>DATE<./+PUBLISHER><'), 'PAGES./-PUBLISHER>/
/                                      /
                                                           /        0   1 54   41 100
```

Figure 1 (Cont.)

```
**DESCRIPTORS**/
/                                                                       /
                                                                          / 55  1 55  21 100
PAGENO/
*EOF
*PRINT
*CHANGE
    COUNTER = 0;
*SCREEN
'SIM' IN DESCR
*HEADING
      COLUMNS = 49, LINES =12, PAGES = 1:
SIMULATION REFERENCES
*FORMAT
/                        /
                                                                     /       3  1 52   1 100
AUTHOR/
/                             /
                                                                             0  1 54  40 100
TITLE<PUBLICATION/-TITLE>/
*EOF
*PRINT   SCREEN
*HEADING
     COLUMNS = 38, LINES = 10,   PAGES = 0:
SIMULATION REFERENCES NOT LOCATED AT GEH
*SCREEN
('SIM' IN DESCRIPTORS AND (NOT 'GEH' IN LOCATION))
*PRINT  SCREEN
*PRINTCOUNTER  REFERENCES MET SCREENING CRITERIA
     COLUMNS =28, LINES = 10, PAGES = 0:
*ENDJOB
```

Figure 1 (Cont.)

in a variety of ways. Though not readily apparent at this point, the input deck can be logically divided into four groupings.

The first such group begins with the *SETUP card and ends with the first appearance of the *EOF card. This group provides GIPSY with the necessary information to set up the original data structure. The *SETUP card and the three cards that follow provide an estimate of the total number of cards that make up the input deck, an estimate of the total number of records in the data file, the maximum number of entries in any one record, and names to be assigned to each entry. For example, entry number one is to be named "AUTHOR," entry number two is to be named "TITLE," etc..

The *DATA card and all the cards that follow down to and including the *EOF card complete this first group. Between these two control cards are the actual reference descriptions that make up the data file. The data punched in columns 77-80 is not included in the data record, but is treated as a special program variable and can be used in a variety of ways, in this example it was used to contain an identification number for each record. Now that the data structure has been set up, the remaining groupings describe information processing tasks to be performed by GIPSY.

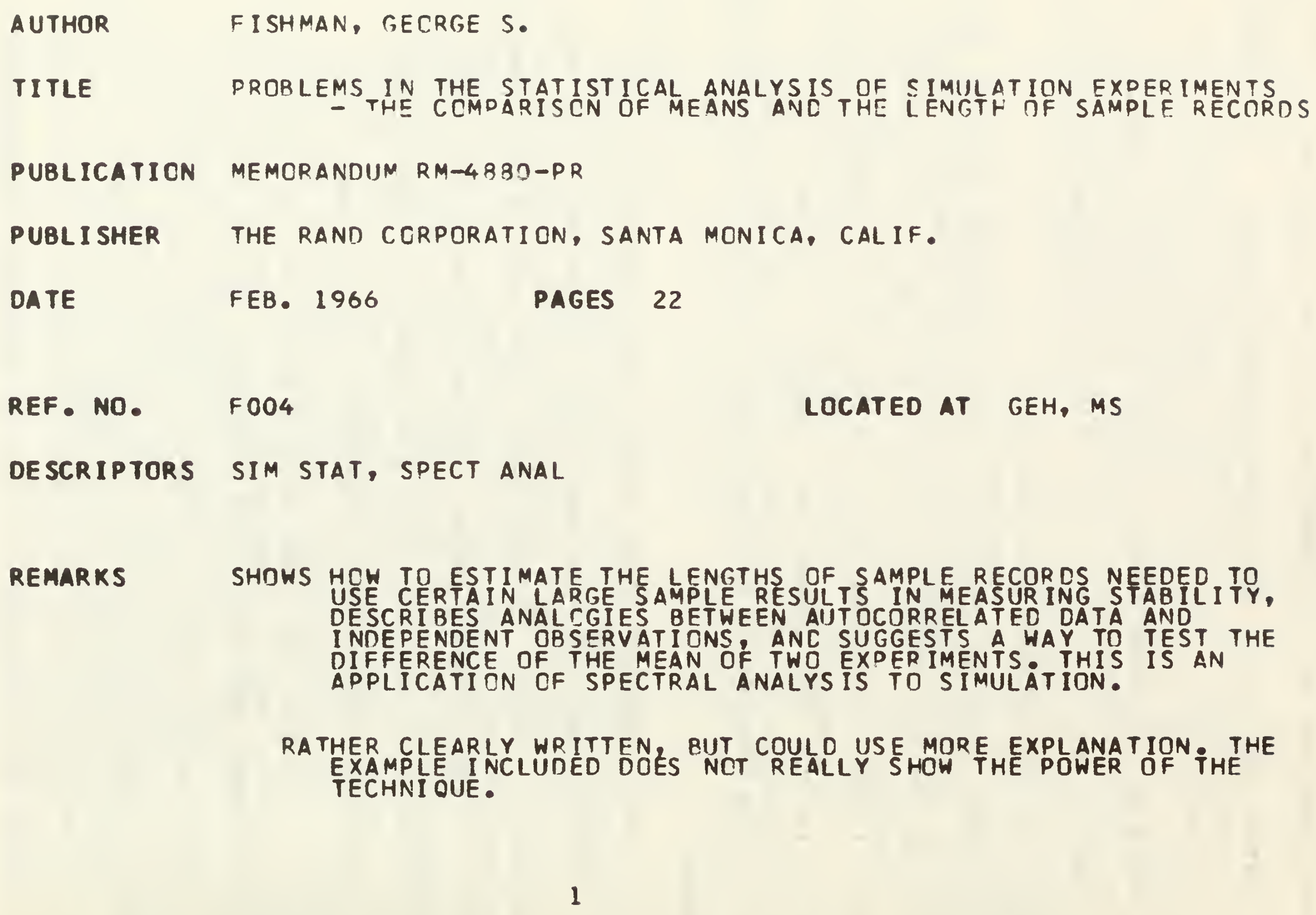
B. PROCESSING THE DATA

The first of these task descriptions includes the cards down to and including the first *PRINT card. The *FORMAT card marks the beginning of a description of the format to be used whenever any information is to be output and the *EOF card marks the end of the format specification. The *PRINT card tells GIPSY to print all the records in the data file according to the most recent format specified. Figures 2, 3, and 4 are copies of the printouts obtained when this section of the program was executed.

The next logical grouping, which continues down to and including the next *PRINT card, is similar to the previous group. It also requires that all records be printed; however, this time the format was changed. Figure 5 is a copy of the results of the execution of this section of the program.

All the remaining cards in the deck except the last one, *ENDJOB, make up the last group. However, this last group can be further divided into three subsections predicated on three related tasks. Before these tasks begin, the *CHANGE card and the card immediately following cause the program variable named "COUNTER," which counts the number of records that have been printed, to be reset to 0.

The screen card marks the beginning of the first task and indicates that the next card contains the screening criteria to be used whenever a screen is indicated. In this case, it would look for the first occurrrence of the character string 'SIM' anywhere in the entry named "DESCRIPTORS." This screening criteria is to remain in effect until changed by a new *SCREEN card or until the program is terminated.



AUTHOR FISHMAN, GECRGE S.

TITLE PROBLEMS IN THE STATISTICAL ANALYSIS OF SIMULATION EXPERIMENTS - THE COMPARISCN OF MEANS AND THE LENGTF OF SAMPLE RECORDS

PUBLICATION MEMORANDUM RM-4880-PR

PUBLISHER THE RAND CORPORATION, SANTA MONICA, CALIF.

DATE FEB. 1966 **PAGES** 22

REF. NO. F004 **LOCATED AT** GEH, MS

DESCRIPTORS SIM STAT, SPECT ANAL

REMARKS SHOWS HOW TO ESTIMATE THE LENGTHS OF SAMPLE RECORDS NEEDED TO USE CERTAIN LARGE SAMPLE RESULTS IN MEASURING STABILITY, DESCRIBES ANALCGIES BETWEEN AUTOCORRELATED DATA AND INDEPENDENT OBSERVATIONS, ANC SUGGESTS A WAY TO TEST THE DIFFERENCE OF THE MEAN OF TWO EXPERIMENTS. THIS IS AN APPLICATION OF SPECTRAL ANALYSIS TO SIMULATION.

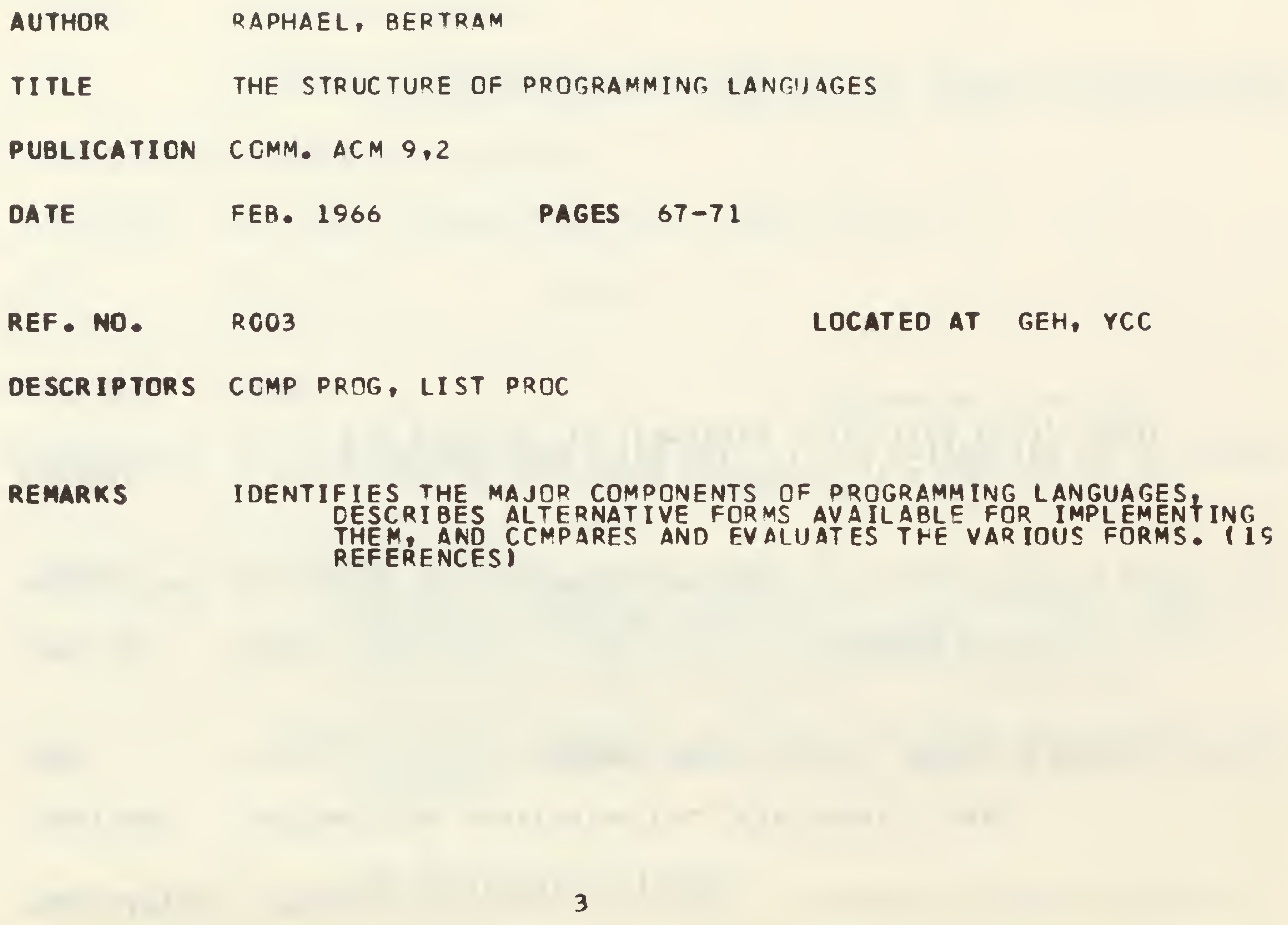
RATHER CLEARLY WRITTEN, BUT COULD USE MORE EXPLANATION. THE EXAMPLE INCLUDED DOES NOT REALLY SHOW THE POWER OF THE TECHNIQUE.

1

Figure 2.

```
AUTHOR        HCGGATT, AUSTIN C., AND BALDERSTON, FREDERICK E. (EDS.)

PUBLICATION   SYMPOSIUM ON SIMULATICN MODELS - METHODS AND APPLICATIONS TO
                   THE BEHAVIORAL SCIENCES

PUBLISHER     SCUTHWESTERN PUBLISHING CO., CINCINNATI, OHIO

DATE          1963              PAGES  289            PRICE  $4.25


REF. NO.      H002                                 LOCATED AT  ESJ

DESCRIPTORS   SIM METH, SIM CCG PRCC, SIM MCD


REMARKS       A COLLECTICN OF 16 PAPERS PRESENTED AT A CONFERENCE ON
                   COMPUTER SIMULATION HELD AT THE UNIVERSITY OF CALIFORNIA,
                   LOS ANGELES, IN SEPTEMBFR 1961. (SEE CC01, H003)




                                    2
```

Figure 3.



```
AUTHOR       RAPHAEL, BERTRAM

TITLE        THE STRUCTURE OF PROGRAMMING LANGUAGES

PUBLICATION  CCMM. ACM 9,2

DATE         FEB. 1966       PAGES  67-71


REF. NO.     RC03                              LOCATED AT  GEH, YCC

DESCRIPTORS  CCMP PROG, LIST PROC


REMARKS      IDENTIFIES THE MAJOR COMPONENTS OF PROGRAMMING LANGUAGES,
                  DESCRIBES ALTERNATIVE FORMS AVAILABLE FOR IMPLEMENTING
                  THEM, AND CCMPARES AND EVALUATES THE VARIOUS FORMS. (19
                  REFERENCES)




                              3
```

Figure 4.

```
                          B I B L I O G R A P H Y

1. FISHMAN, GEORGE S., "PROBLEMS IN THE STATISTICAL ANALYSIS OF
     SIMULATION EXPERIMENTS - THE COMPARISON OF MEANS AND THE LENGTH OF
     SAMPLE RECORDS," MEMORANDUM RM-4880-PR, THE RAND CORPORATION, SANTA
     MONICA, CALIF.,FEB. 1966.
       **SIM STAT, SPECT ANAL**


2. HOGGATT, AUSTIN C., AND BALDERSTON, FREDERICK E. (EDS.), SYMPOSIUM ON
     SIMULATION MODELS - METHODS AND APPLICATIONS TO THE BEHAVIORAL
     SCIENCES, SOUTHWESTERN PUBLISHING CO., CINCINNATI, OHIO,1963.
       **SIM METH, SIM COG PROC, SIM MOD**


3. RAPHAEL, BERTRAM, "THE STRUCTURE OF PROGRAMMING LANGUAGES," COMM. ACM
     9,2,(FEB. 1966), 67-71.
       **COMP PROG, LIST PROC**





                                    1
```

Figure 5.

Next the *HEADING card causes a new page to be ejected by the printer and the heading "SIMULATION REFERENCES" to be printed starting in column 49 of the twelfth line down from the top of the page. Next the *FORMAT card marks the beginning of new format specifications for subsequent printings. Finally the *PRINT SCREEN card tells GIPSY to print only those records that satisfy the prevailing screening criteria and print them according to the latest format information provided.

The second task begins with a new *HEADING card which causes 10 lines on the current page to be skipped and the text "SIMULATION REFERENCES NOT LOCATED AT GEH" to be printed beginning in column 38. Next, new screening criteria is encountered which replaces the previous one. Once again a *PRINT SCREEN card causes the records to be subjected to the prevailing screen and the ones that satisfy it to be printed. Note that a new format was not specified, therefore, the records that passed this latest screen will be printed in exactly the same format as were the ones in the previous section.

The last task in this group, the *PRINTCOUNTER card, causes 10 lines on the current page to be skipped and the number of records that satisfactorily passed both of the screens to be printed starting in column 28 followed immediately by the text "REFERENCES MET SCREENING CRITERIA." Figure 6 is a copy of the results of the execution of these three tasks.

```
                              SIMULATION REFERENCES

FISHMAN, GEORGE S.
     PROBLEMS IN THE STATISTICAL ANALYSIS OF SIMULATION EXPERIMENTS - THE
                COMPARISON OF MEANS AND THE LENGTH OF SAMPLE RECORDS


HOGGATT, AUSTIN C., AND BALDERSTON, FREDERICK E. (EDS.)
     SYMPOSIUM ON SIMULATION MODELS - METHODS AND APPLICATIONS TO THE
                BEHAVIORAL SCIENCES




                  SIMULATION REFERENCES NOT LOCATED AT GEH


HOGGATT, AUSTIN C., AND BALDERSTON, FREDERICK E. (EDS.)
     SYMPOSIUM ON SIMULATION MODELS - METHODS AND APPLICATIONS TO THE
                BEHAVIORAL SCIENCES




        3 REFERENCES MET SCREENING CRITERIA
```

Figure 6

Finally, the last card in the deck, *ENDJOB, causes the printer to eject a new page and print the message "END OF JOB" across the top of the new page.

The total amount of computer time required to produce Figures 2 through 6, utilizing the data deck as it appears in Figure 1, was 1.5 seconds. A certain amount of this time is fixed overhead, and computer time would not increase proportionally as the data base and/or the number of tasks are increased. For example, when the data file was expanded to 222 references and was processed according to these same requirements, the computer time expended was 73.6 seconds.

C. THE BIBLIOGRAPHY DATA FORM

Figure 7 is an example of a form that was used in the preparation of the data for the bibliography mentioned above. Once the user has decided upon his file organization a form similar to Figure 7 can be reproduced inexpensively. This form can then be used during data accumulation and can then be used to keypunch the data onto cards.

AUTHOR Fishman, George S. /

TITLE Problems in the Statistical Analysis of Simulation Experiments - The Comparison of Means and Length of Sample Records /

PUBLICATION Memorandum RM-4880-PR /

PUBLISHER The RAND Corporation, Santa Monica, Calif. /

DATE Feb. 1966 / PAGES 22 / PRICE /

REF. NO. F004 / CITED IN / LOCATED AT GEH, MS /

DESCRIPTORS Sim Stat, Spect Anal /

REMARKS Shows how to estimate the lengths of sample records needed to use certain large sample results in measuring stability, describes analogies between autocorrelated data and independent observations, and suggests a way to test the difference of the mean of two experiments, this is an application of spectral analysis to simulation. /

Rather clearly written, but could use more explanations. The example included does not really show the power of the technique. /

Figure 7.

III. THE INPUT DECK

As can be seen from the example in Figure 1, an input deck for GIPSY consists of numerous control cards (with a star in column 1) interspersed with the data to be processed and other information. The typical order of input is to name the entries and then furnish the data to be processed. Next, specify a format and maybe a screen and then produce some output. Then specify either a new format or a new screen or both and then produce more output. This may be done any number of times. Also, cards may be included to print headings and to control various program variables. It is also possible to enter a new batch of data, with or without renaming the entries.

A. OS/360 REQUIREMENTS

To process an input deck to GIPSY on the computer at the Naval Postgraduate School, various OS/360 control cards are required. If the program is to be run utilizing a source deck, then the following are the control cards that are needed:

```
// (User's Green Job Card)
// EXEC PL1LFCLG,REGION.GO=xxxK,TIME.GO=xx
//PL1L.SYSIN DD *
(Source Deck)
//GO.SYSIN DD *
(User's Input Deck)
/* (Salmon Colored)
```

To process an input deck utilizing a source deck requires that the program be compiled and link edited each time that it is run. These two steps require approximately 50 seconds of computer time. To save this unnecessary use of computer time and also to spare the user the inconvenience of repeated handling of a large source deck, GIPSY has been compiled, linked edited, and stored, as an object module, on the IBM 2314 disk named "MARY." As a result, the only computer time required is that needed to process the input deck and the only cards required are those that make up an input deck to GIPSY. The following are the control cards needed to process an input deck utilizing the program stored on the disk:

```
// (USERS GREEN JOB CARD)
//JOBLIB DD DSN=F0849.GIPSY,VOL=SER=MARY,UNIT=2314,DISP=OLD
// EXEC PGM=GIPSY,REGION=xxxK,TIME=xx
//SYSPRINT DD SYSOUT=A,SPACE=(TRK,(xxx,xxx),RLSE),
// DCB=(RECFM=FB,LRECL=133,BLKSIZE=3325)
//SYSIN DD *
(USERS INPUT DECK)
/* (SALMON COLORED)
```

The x's in the above cards represent the time and storage requirements of the program and the input deck. These parameters will vary depending on the application. The region size may be <u>approximated</u> as follows: GIPSY=150K plus 32K for each allocation of WORD(i). (See sections III-B and VIII-B for an explanation of WORD.)

B. SETTING UP THE PROGRAM

The first part of the input deck sort of "sets the stage" by providing GIPSY with certain information necessary to set up and get ready to process the data. Thus, it is mandatory that the first card in the input deck be a card with *SETUP punched in columns 1-6 (unless just the listing described in Section III-D is desired).

There are eight program variables used to allocate storage and to dimension other variables. The user furnishes GIPSY with a rough description of his data in order that efficient use of storage can be effected. The following is a description of these variables.

CARDS - This is to be an estimate of the total number of cards in the input deck. Default value is 480.

COLS - This is an estimate of the average number of columns used per card to furnish the data in the input deck. Default value is 70. This variable is multiplied by CARDS in the program to arrive at the total number of bytes to allocate to the program. Thus, this sets an upper limit on the size of the user's data deck.

ESTIMATE - This variable is used for dimensioning purposes and is an estimate of the number of data records in the data file. Default value is 100.

This variable sets an upper limit on the number of records in a data file.

ENTITY - The value assigned to this variable is the actual number of entries that make up a data record. Default value is 10.

NOLINES - This gives the maximum number of logical lines to be described in any format specification. Default value is the value of ENTITY; (see Section V).

IDCOL - The value assigned to this variable is the column number in which the special field on the input cards is to begin. This variable is further described in Sections IV-B. Default falue is 77.

SIZE - This variable specifies the field width desired to output the value of COUNTER. Default value is 3. COUNTER is always an integer, right justified in this field.

COLOR - This variable specifies the number of times an item is to be printed when over-printing is specified in an output format. Default value is 3.

The value assigned to each of these variables must be a decimal integer. The card containing these variables and their values must follow the *SETUP card. The format for this card is: VARIABLE=VALUE, VARIABLE=VALUE, etc.; and may be continued on as many cards as is necessary, with the variables and their values being separated by commas and a semicolon terminating the list. If no values are assigned and default values are desired then a card with a semicolon punched anywhere must be included at this point in the input deck.

C. NAMING THE ENTRIES

The final requirement of the *SETUP card is that names must be given to each of the entries that describe a data record (author, title, publication, etc. in the bibliography example). These names may then be used to refer to entries when specifying a screen, outputting information, etc.. Entry names are restricted to being alphanumeric character strings of length less than or equal to 50 characters. That is, names cannot contain blanks or special characters, and the first character of the name must be an alphabetical character. A further restriction is that entry names may not be any of the following: IN, AT, GT, LT, GE, LE, EQ, NE, OR, AND, NOT. Entry names are punched with the first name beginning in column 1 of the first card, and are separated by slashes. Note that blanks may not appear between slashes as they would be interpreted as being part of that character string and blanks are not permissable characters.

No characters may be punched in or after the column specified by IDCOL on the previous card. If the list of names will not fit on one card, as many continuation cards may be used as is necessary. The list may be broken between any two entry names, immediately after the slash. The first character of the next name begins in column 6 of the next card, and so on until the list has been completed. These cards are to immediately follow the cards described above, in the input deck. The reader is referred to Figure 1 for an example of how the *SETUP group of cards should look.

D. INPUT DECK LISTING

GIPSY has a limited capability of providing the user with a listing of his input deck. To cause a listing to be printed, a card may be inserted with *LIST punched in columns 1-5 between the JCL cards and the first card in the input deck. The result will be that all cards in the input stream, beginning with the one immediately following the *LIST card and ending with the one immediately in front of the salmon colored /* card, will be printed. The program will then terminate normally, without any further processing.

IV. THE DATA TO BE PROCESSED

The data to be processed would normally be the greatest part of the input deck. The only limit on the number of records that can be processed is the amount of core storage available to the user. Each data record consists of values for the entries named in the *SETUP section. The data would normally follow the *SETUP group and is preceded by a control card with *DATA punched in columns 1-5. Another control card with *EOF punched in columns 1-4 is required to mark the end of the data file.

A. THE DATA CARDS

A data record is punched on a set of cards. Since the only limitation on the size of a record is that its total length be less than 3000 characters, as many cards as are needed may be used, with the data beginning in column 1 of the first card and in column 6 of all others. Data should not be punched in or after the column specified by IDCOL (to be discussed later in this section).

Entries are separated by slashes. Consequently, all characters appearing between two slashes are taken as being constituents of that entry, the only exception being when breaking an entry for continuation on the next card. If the last character punched on the card is a slash, then no blanks are assumed and whatever appears in column 6 of the next card is treated as the first character of the next entry. If, on the other hand, the break occurs in the middle of an entry, one blank is assumed between the last non-blank character on the card and the character in column 6 of the next card.

There must be a one-to-one correspondence between the entries named in the *SETUP section and the entries in a data record. To maintain this correspondence, when a record contains a null entry, a slash is punched in the next column following the slash that marks the end of the previous entry. If these null entries occur at the end of the data record then the slashes may be omitted since GIPSY will automatically regard them as null.

If the first two entries in a data record were null, then slashes would appear in columns 1 and 2. However, this would result in the computer interpreting that card as a OS/360 JCL card, and the program would terminate. One way to avoid this problem is to assign a blank to either one of these entries.

B. THE "IDENT" FIELD

Column IDCOL to column 80 on the first card of every data record is handled specially. The contents of this field are stored in a character variable of length (80-IDCOL) + 1 and may be accessed by the user through the variable IDENT. In the bibliography example, this field was used to contain an identification number for each record and IDCOL had the value 77 (by default). Of course, as this field becomes larger it reduces the number of columns that may be used for data on every card.

V. THE OUTPUT FORMAT

The format of the output from GIPSY is very flexible, but its specification is the most difficult part of the input deck to prepare. Fortunately, there usually would be only a few standard formats of interest in any particular application of the system, and these would have to be made up only once. Before proceeding, the difference between a "locical" line of output and a "physical" line of output should be understood. A "physical" line consists of a maximum of 132 characters and is one actual line produced by the line printer. A "logical" line is a line as specified by the user and may contain so many characters that it requires several "physical" lines to actually print it.

A complete format specification describes one or more logical lines, and the description of each logical line is contained on two or more punched cards. A control card with *FORMAT punched in columns 1-7 marks the beginning of a format specification, and a control card with *EOF punched in columns 1-4 marks the end.

The specification of a logical line can be thought of as having two components, the form of the line and the content of the line. The form of the line is required for every logical line and is described on two cards hereafter referred to as form cards. The content of the line is described on one or more cards hereafter referred to as content cards.

A. FORM OF THE LINE

It is convenient for the discussion if the two form cards are envisioned as being placed end-to-end, next to each other so that they make up a field of 160 columns. The first 133 columns of this field (all of the first card and columns 1-53 of the second) then represent the 133 print positions on the line printer. The remaining 27 columns are divided into numerous small fields that will be described in Section B below.

A slash must always be punched in column one of the first form card in each logical line, the remaining 132 columns are then available to the user. These 132 columns represent print positions and may be filled by three types of characters: blanks, text, and variables. Text is written into the print positions where it is desired to have it appear. Slashes may not be used in the text since they are used to indicate the first print position for a variable which is defined on the associated content cards. Blanks are used as text to be printed or as spacers for the variables.

Since a slash marks the print position for the first character of the variable, it must be followed by a sufficient number of blanks to allow the placement of the value of the variable in that space, before the occurrence of the next slash or the next character of text. If the variable called for is too long to fit on the remainder of the physical line, continuation is provided for as will be described shortly.

B. FORMAT PARAMETER FIELDS

The remaining 27 columns of the second form card for each logical line must contain the following information:

Column 55: A *slash* in this column indicates that there is no content card associated with this logical line. A *blank* in this column indicates that the next card in the input deck is the content card that defines the variables provided for on these form cards.

Column 56: A *slash* in this column indicates that when a variable called for in this logical line evaluates to the null string, printing of all text since the processing of the previous variable and any associated text on the content card is to be suppressed. A *blank* indicates that there is to be no suppression of any printing in this logical line.

Column 57: A *slash* indicates that each time this logical line is processed the resulting printing is to begin on a new page. A *blank* indicates that continuation on the same page is desired.

Column 59: A *slash* indicates that this logical line is a page heading and is to be printed, as described, each time a new page is ejected.

This logical line must be less than or equal to one physical line and variables cannot be included. As many page heading lines may be indicated as is desired. A blank indicates that this logical line is not a page heading.

Column 61: A slash indicates that all text appearing in this logical line is to be over-printed. The result is that the text will be in much darker print than the variables. A blank indicates that no over-printing is desired.

Column 63-64: A decimal integer, right justified, must be supplied which indicates the number of blank lines that are to be inserted between the last line of the previous logical line printed (or the top of the page) and the first line printed from this logical line. This field cannot be left blank.

Columns 69-70: A decimal integer, right justified, must be supplied which indicates the last physical line on the page on which printing from this logical line may take place, and after which no printing is to take place. There can be a maximum of 59 physical lines on a page. This field cannot be left blank.

Columns 72-74: A decimal integer, right justified, must be supplied which indicates the column in which printing is to begin if additional physical lines are required to contain this logical line (indentation for continuation lines). This field cannot be left blank.

Columns 76-78: A decimal integer, right justified, which indicates the column number after which no printing is to take place for all physical lines in this logical line. This field cannot be left blank.

Column 80: A _slash_ indicates that this logical line is to be used only once, the first time the printing section is entered (for example, a title page on a report). This logical line must be less than or equal to one physical line and variables cannot be included. As many logical lines of this type may be specified as is desired. A _blank_ indicates that this logical line is not to be used in this manner.

C. CONTENT OF THE LINE

The content cards, associated with a logical line and its associated form cards, are used to supply the variables that are to be processed with the logical line. A content card may be thought of as being divided into fields. Within each field there may appear literals, variables, and/or conditional expressions. To each slash on the form card (in a print position) there must correspond a field on the content card. However, any number of fields may be concatenated to form a new field.

The basic ingredient in any field is a variable which may be any entry name or one of the program variables: IDENT, COUNTER, or PAGENO. Entry names and IDENT have been previously discussed, COUNTER and PAGENO will be defined at this point before continuing with the description of the fields.

In GIPSY there is a variable named COUNTER which is incremented by one each time a data record is processed for printing. COUNTER is set to zero each time the *SETUP and the *PRINTCOUNTER control cards are processed. COUNTER may be set to any value by the user utilizing the *CHANGE control card (see Section VII-A). COUNTER can be used in two ways: to count the number of records that successfully pass a given screen and/or to number the records as they are printed. PAGENO is the GIPSY variable that keeps track of the page numbers of pages that are printed. PAGENO is initialized each time the program processes the *PRINT control card. All pages printed are counted, except the pages that are used in a manner similar to a title page (a slash in column 80 of the second format card). Pages are counted even though page numbering is not called for.

Literals may be defined as any character that appears in the field and it is not one of the variables defined above or it is not a conditional expression. It is especially efficient if the literals are enclosed in single quotes. In the case of alphanumeric literal strings, it is imperative that they be enclosed in single quotes.

Example:

```
         Literals
       ┌───┬─────┬───────────────┐
       ↓   ↓     ↓               ↓
COUNTER./  "TITLE"//', NUMBER OF PAGES = 'PAGES/
   ↑          ↑                              ↑
   └──────────┴──────
                Variables
```

If the fourth record was being processed, and its title was "SIMULATION" and it contained 34 pages, then the following output would be produced: 4. "SIMULATION," NUMBER OF PAGES = 34.

A conditional expression is enclosed in angular brackets and, in general terms, says to print certain information only if a certain condition is satisfied. Inside the angular brackets the information to be printed is separated from the condition by a slash. Literals and variables, as defined above, may appear between the left angular bracket and the slash. They will be printed only if the condition defined to the right of the slash is satisfied. Immediately following the slash must appear either a "+" or a "-" followed immediately by a variable name, as defined above, followed immediately by the right angular bracket. The "+" is interpreted as "if the following variable is not null," the "-" is interpreted as "if the following variable is null." Roughly speaking, a "+" says if something is in

the variable and the "-" says if nothing is in the variable.
Example:

```
        entry names
       ↓           ↓
a.  <PUBLICATION/-TITLE>
```

This example says to print the contents of the entry named "PUBLICATION" if the contents of the entry named "TITLE" is the null string.

```
   literals   entry name
       ↓           ↓
b.  <'PAGES = '/+PAGES>
```

This example says to print the character string 'PAGES =' if the contents of the entry named "PAGES" is <u>not</u> the null string.

Fields are separated by slashes. Two fields may be concatenated by separating them by two slashes instead of one slash. This merges the two fields into one field which has a corresponding slash in a print position on the associated form card. Content cards are punched with the first field beginning in column one of the first content card and must end before column IDCOL. If the content line is too long to fit on one card, it may be continued by breaking the string immediately after a slash and punching the next character in column six of the next card. This process is repeated until the entire content line has been specified. Example:

```
    COUNTER./ AUTHOR,// "TITLE,"// PUBLICATION,// PUBLISHER,//
  <(/-PUBLISHER>DATE<./+PUBLISHER><'), 'PAGES./ -PUBLISHER >/
```

This example says to print the value of the counter, followed by a period, in the column indicated by the first slash on the form card. Then in the next print position marked by a slash on the form card, print the author's name followed by the title and a comma, all in double quotes, concatenated with a blank followed by the name of the publication and a comma, concatenated with a blank followed by the publisher and a comma. Next, the date is to be concatenated. If the publisher is null, then the date is to be parenthesized followed by a comma, a blank, the number of pages and a period. If the publisher is not null, then the date is simply to be followed by a period. Figure 5 is the output produced by this example.

D. ORGANIZATION OF THE FORMAT SPECIFICATIONS

The organization of the format specifications requires that any form cards that are to specify a title page must be the first cards immediately following the *FORMAT card. Next must come any form cards that specify a page heading. All cards that come after this point are cycled through for each record processed for printing. Therefore, next would come the first and second form cards and then the content card for the first logical line. Next would come the cards for the second logical line, and so on until the entire format has been specified. The only limit on the number of logical lines in a format specification is the value assigned to the variable NOLINES in the *SETUP section.

If page numbering is desired then its format must be provided for by the user. The variable PAGENO described above is indicated on the associated content card. Any text may appear on the form cards. However, PAGENO is the only variable that can be specified, and the length of the logical line must be less than or equal to one physical line. Page numbering can only be accomplished as the last printing to take place on a page. The decimal integer specified in columns 63 and 64 on the second format card of a page numbering specification is interpreted as the number of the line on the page that the page number is to appear. This number must be less than or equal to the number in columns 69 and 70. The page number logical line may occur anywhere in the format specification.

VI. SCREENING

A screen in GIPSY is a boolean expression which states a condition that a data record must satisfy in order to be included in the output. This feature is the key to the usefulness of a system like this, in that it makes it easy to obtain listings of various cross-sections of the data being processed, such as all references on a specific subject in a bibliography or all members with a certain attribute in a membership list. A screen specification consists of a *SCREEN card followed by one or more cards with a boolean expression.

A. SIMPLE BOOLEAN EXPRESSIONS

A simple boolean expression is of the form: 'character string' relational operator entry name. The character string can be any group of characters, including blanks but not including single quotes. It is the character string that will be tested for according to the type of relation. The entry name can be any of those given in the *SETUP section or an abbreviation of one, made up from its leading characters. If an abbreviation is used, it must be long enough to distinguish it from any similar name that appears before it in the list of names. The relational operators and their description are as follows:

IN - The IN expression has the value true when the character string occurs anywhere in the designated entry of a data record.

AT - The AT expression has the value true only when the character string occurs *at the beginning* of the designated entry.

GT - The GT expression has the value true when the character string yields a result of *strictly greater than* when compared character by character with the entry designated.

LT - The LT expression has the value true when the character string yields a result of *strictly less than* when compared character by character with the entry designated.

GE - The GE expression has the value true when the character string yields a result of *greater than or equal* when compared character by character with the entry designated.

LE - The LE expression has the value true when the character string yields a result of *less than or equal* when compared character by character with the entry designated.

EQ - The EQ expression has the value true when the character string yields a result of *equal* when compared character by character with the entry designated.

NE - The NE expression has the value true when the character string yields the result of *unequal* when compared character by character with the entry designated.

The following are examples of simple boolean expressions:

'SMITH' AT AUTHOR

'SIM' IN TITLE

'1966' LE DATE

B. COMPLEX BOOLEAN EXPRESSIONS

The logical operations AND, OR, and NOT may be used to form "complex boolean expressions," in the usual fashion. The normal hierarchy of boolean operations is assumed, NOT has the highest precedence, AND is next highest, and OR has the lowest precedence. Parentheses may be used to any level desired for grouping or factoring of the expression, the only requirement being that parentheses must be balanced. For example, the expression A AND B OR NOT C AND D would be evaluated as follows: ((A AND B) OR ((NOT C) AND D)). If adjacent simple expressions have the same "relational operator entry name" part, it need be given only once with the last one. Whenever a relational operator appears, it applies to all character strings occurring since the previous relational operator, without regard for parentheses. The following is an example of a boolean expression which might be used with the bibliography in the example:

```
('SIM PROG' OR 'LIST PROC') AND 'LANG COMP' IN DESCRIPTORS
      AND ('1964' OR '1965' OR '1966' IN DATE) AND NOT
      ('GEH' OR 'YCC') AT LOCATION
```

Only those references dealing with the comparison of simulation or list processing languages, published between 1964 and 1966, and not located at GEH or YCC would satisfy this screening condition.

C. THE BOOLEAN EXPRESSION CARDS

A boolean expression is punched on one or more cards and put behind a *SCREEN card in the input deck. As many cards as needed may be used, with the expression beginning in column one of the first card and continuing in column six of the others. Columns IDCOL to Column 80 cannot be used on any of the cards as they are ignored by the program. The expression can be broken anywhere except in the middle of a character string, operator, or an entry name and the program will automatically continue the expression properly. Blanks, except where they appear within single quotes, are not needed and they are ignored by the program.

When a boolean expression is being evaluated for a data record, the scan proceeds from left to right, consequently, some savings in computer time may be realized if, when constructing the screening expression, those elements which are most likely to be true in an OR subexpression or those most likely to be false in an AND subexpression are placed as far to the left as possible.

The only restrictions on the size of the screening expressions are: (a) the total length of the expression cannot exceed 3000 characters, (b) the maximum length of a character string (between single quotes) is 25 characters, and (c) the maximum number of character strings in the expression is 50.

VII. OTHER CONTROL CARDS

Thus far, the uses of five control cards have been explained. These were the *SETUP, *LIST, *DATA, *FORMAT, and *SCREEN, which essentially are required for the basic operation of the system. There are eight other control cards which may be included in an input deck also, to accomplish such things as printing formatted output, printing headings, changing the value of program variables, printing the elapsed computer time, reordering the data file, and signalling the end of a job.

A. *CHANGE (CHANGE VARIABLE'S VALUE)

When this control card is encountered in the input deck the program executes a GET DATA statement which will read the next card in the input deck and all subsequent cards until the variable list is completed (marked by the appearance of a semicolon). The execution of this statement will change the value of any program variable whose name appears in the variable list to the value given. However, it is strongly recommended that the use of this facility be limited to five variables. Four of these were described in the *SETUP section, they are: COUNTER, IDCOL, COLOR, and SIZE. The fifth variable is SLASH.

SLASH is a special program variable and as such is reserved for program use only. This variable is initialized to a '/' when the program is loaded. If the user has a great need to use the character '/', then he may change the value of SLASH to any other character which will then, in turn, be reserved from future use until changed again or the program is terminated. This new character would then be used in lieu of the slash wherever described in this paper. To accomplish the desired changes, one or more cards must follow the *CHANGE card with statements of the form "variable = value." Each of these statements is separated from the next by a comma, and cannot be split between cards. All 80 columns may be used and blanks are ignored, the only other requirement is that the variable list be terminated by a semicolon. All numbers assigned as values must be integers and any character assigned to slash must be enclosed in single quotes.

B. *ORDER (REORDER DATA RECORDS)

Whenever the *DATA control card is encountered in the input deck and the subsequent data file processed, the records in this data file are ordered sequentially as they are read in, and it is in this order that the data records are processed and printed whenever so indicated. This order may be changed at any time by the user, and this new order remains in effect until another change is called for or until termination of the program. To change this order, a card with *ORDER punched in columns 1-6, followed by two or more blanks, followed by the word "BY," followed by zero or more blanks, followed by an entry name (as specified in the *SETUP section), is inserted in the input deck at the point in the processing at which reordering is desired. This ordering procedure reorders the records based on their relative position in the collating sequence, from the lowest to the highest. Null entries are placed at the end of the new order, in the order in which they appear in the input deck. For example, *ORDER BY TITLE would reorder the data file alphabetically by titles.

C. *PRINT (PRINT OUTPUT)

When this control card is encountered in the input deck it causes some information from the data file to be printed. If the records are to be screened (according to the most recent criteria) for inclusion in the output then the word "SCREEN" must appear somewhere on the *PRINT control card. If screening is not be performed and all records are to be included in the output, then the word "SCREEN" must not appear on the *PRINT control card.

The order in which the records are printed is specified by the user. If an *ORDER control card has not appeared previously in the input deck, then the records will be processed in the same order that they appeared in the data deck.

When a record is processed for printing, the most recent format specification is used. There are no default format specifications. If printing is requested before a *FORMAT control card has been processed, an error condition will be raised.

D. *EOF (END OF FILE)

The *EOF control card is only used in two different situations in the input deck. It is used to mark the end of the data file on input and to mark the end of the format specifications. Thus a card with *EOF punched in columns 1-4 should be the last card in the *DATA and the *FORMAT sections of the input deck.

E. *ENDJOB (END OF JOB)

This card is most conveniently placed at the end of an input deck, although it may be used at the ends of complete sections within an input deck. It merely causes the message "END OF JOB" to be printed at the top of the next page in the output, and does not actually cause termination of the program.

The remainder of the control cards have one requirement in common, and it will be described at this point instead of separately for each control card. These control cards when encountered in the input deck cause some printing to take place. Thus, the one thing they have in common is the card which is to provide the necessary information to format the output. This card is of the following form: COLUMNS = X, LINES = Y, PAGES = Z;. Where X is the number of the column in which the specified printing is to begin, Y is the number of lines that are to be skipped down the page and on which the printing is to begin, and Z is the number of pages that are to be ejected before printing is to begin. This card may be punched in a free format as long as a comma separates the items in the list and a semicolon marks the end of the list. All three of these variables have a default value of one, thus they need only be specified when a different value is desired. If all default values are desired, a card must then be inserted with a semicolon punched anywhere in columns 1 to 80.

F. *HEADING (PRINT HEADING)

The *HEADING card provides the user with the capability of inserting formatted text, which is independent of the data being processed, in the output stream. Three cards are required to print a heading, the *HEADING card followed by the formatting card described above, followed by a card with the text to be printed, punched starting in column one. All 80 columns of this card are printed.

G. *PRINTCOUNTER (PRINT "COUNTER")

This control card is associated with the program variable COUNTER. As mentioned earlier, COUNTER counts the data records that are passed to the printing section of the program. When *PRINTCOUNTER is encountered in the input stream, it immediately reads the next card which contains the formatting information described above. The text is taken from columns 15 to 80 of the *PRINTCOUNTER card and is concatenated with the value of COUNTER. This string is printed according to the format specified. Then COUNTER is reset to zero.

H. *TIME (PRINT ELAPSED TIME)

This control card enables the user to determine the computer time taken to perform various tasks with GIPSY. When it is encountered, the elapsed time, in seconds, since the previous *TIME card was processed is printed, followed by the contents of columns 15 to 80 of this card. A second card must follow the *TIME card with the formatting information described above. For the first *TIME card in the deck, the elapsed time is computed from the time the progam was loaded.

For these elapsed time computations, the current time of day is used. Thus, unfortunately, when GIPSY is being run in an MVT environment, the elapsed time would be a total elapsed time, including OS/360 interrupts, and not just the amount of CPU time required by GIPSY.

VII. SYSTEM IMPLEMENTATION

Basically GIPSY is organized into separate sections of computer code, each of which performs a different information processing task. This program organization, made possible by the ability to subscript labels in PL/I, was chosen over a "procedure" or "subroutine" orientation in that it would be faster, while requiring approximately the same amount of coding. Entry into (and departure from) a segement of code is caused by the occurrence of a control card in the input stream. As a consequence, an input deck to GIPSY is divided into sections according to the tasks to be performed, with each section preceded by a control card.

The names of the control cards that are associated with eleven such sections of the program are placed in the array, CONTROL(i). When a control card is encountered, its name is looked up in CONTROL(i) and this subscript is then used to provide the proper subscript for the label variable SEGMENT(i) when a GOTO SEGMENT(i) statement is executed. The control card *SETUP is handled a little differently in that instead of returning its position in the array CONTROL, a statement GOTO SETUPS is executed, the reason for this variation will become clear later.

A. SETTING UP THE PROGRAM

The first executable section of code in the program fetches the current time of day, converts it from a character variable which gives the time in hours, minutes, seconds and milliseconds into a fixed binary variable which gives the time in milliseconds, and then stores this value in a variable named CLOCK. This would be used later, the first time the *TIME control card is processed. This is followed immediately be a section that processes the *LIST control card if it is present as the very first card of the input deck (after the JCL cards).

The *LIST section is very simple in that it reads a card from the input stream and immediately outputs this same card, centering it on the output page. It starts printing the card images on line 13 of the page and then prints 46 such images, ejects a page, prints another page, and so on until the input stream is emptied. At this point control is transferred to the end of the program and termination occurs normally.

If on the other hand, the first card encountered in the input deck is the *SETUP control card the program begins setting up to process the input deck. The first thing GIPSY does is to assign default values to several variables. A GET DATA statement is then executed to read in the next card, and any continuation cards, obtaining the user supplied values.

At this point in the execution of the program, utilizing the quantities just obtained (or their default values), calculations are made to determine the amount of storage to request from OS/360 and to determine the dimensions of

19 subscripted variables. Next the internally nested procedure labeled START is called and these 19 variables are declared utilizing the results of the calculations just described. The method just described makes use of the capability in PL/I to pass the values of variables to subordinate procedures for dimensioning the locally declared variables, to allocate storage dynamically. In this manner the storage allocated more closely approximates the needs of the application and thus is more efficient than a blind, fixed allocation.

Before the last bit of setting up is done, a few variables are initialized. Finally then, the procedure PACKHOLD, which extracts character strings from cards, packing them into the variable HOLD, is called to read in the remaining data cards in the *SETUP section. The variable HOLD now contains all the entry names, separated by slashes. The program then processes HOLD, picking off the entry names and placing them in the array, ATTRIBUTES(i). (ATTRIBUTES(i) and CONTROL(i) are the only two tables which ever require a lookup during the execution of the entire program).

After the last card naming the entries has been read, the next card in the input deck is read and a look up for a control card is conducted, if not found, a diagnostic is printed and the program execution terminated; otherwise a branch is made to that section of the program which processes the control card found.

B. PROCESSING THE DATA FILE

The next control card normally found at this point is the *DATA card. This card marks the beginning of the user's data file that is to be stored at this point and then manipulated throughout the remainder of the execution of GIPSY. This data base is stored in an array of sequentially ordered lists named WORD(i). It will facilitate understanding of the remainder of this section if the reader will refer to Figure 8 freely. WORD is a character variable with a length attribute of 32767 characters, varying. This is obviously one of the variables whose dimension is calculated with user supplied information.

When PL/I allocates storage for a variable with a varying length attribute it has no choice but to reserve space for its maximum length. Therefore, storage is allocated for the data file in chunks of 32K bytes of core. The calculation of the dimension of WORD (how many 32K chunks the user needs) is done as follows:

```
STORAGE = CARDS * COLS:
L = (STORAGE/ 32767) + 1;
```

WORD is then given the dimension L each time the Procedure START is entered as the result of the occurrence of a *SETUP control card.

TABLE(i,j) is another important variable necessary for the storage of the data file which is dimensioned with user supplied quantities. In the declaration statement i = ESTIMATE and j = ENTITY + 2 (entity is the number of entries in a data record). The rows of TABLE correspond to data records, and the columns correspond to entries. To input and store the data file the PACKHOLD procedure is called which obtains the first data record, packs it into HOLD, and then returns HOLD. The program then places the record (without the slashes), character by character into contiguous character positions of WORD. As it does this, it also places pointers to the beginning character position for each entry into TABLE(i,j).

After the record is processed, the data found in the IDENT field is placed in the array SERIALS(i), the pointer to the next available character position in WORD is placed in TABLE(i, ENTITY+1), the value of the current subscript of WORD is placed in TABLE(i, ENTITY+2), and then i (the row number) is incremented by one. This process is repeated over and over until the *EOF control card is encountered in the input stream.

C. THE FORMAT SPECIFICATIONS

To process the format specifications requires 19 one-dimensional arrays. These arrays fall into two groups, based on how their subscripts are derived. In one group, the subscript is the number of the logical line that the particular value is associated with. This group can be thought of as making up one large format table, where the rows correspond to the logical lines and the columns are the various one-dimensional arrays. In the other group,

the subscript is simply the number of the occurrence of that particular item since the beginning of the format specifications. The reader will find it helpful to refer to Figure 9 while reading the remainder of this section.

The reason that so many arrays are required is that the format specifications are preprocessed to the maximum extent that is feasible, the philosophy being that the extra time and space expended at this point greatly enhances overall program efficiency since this processing is done only once and each format specification is processed many, many times during the output operations. This section of the program is enclosed in one large loop that is entered when the *FORMAT control card is encountered. It processes the format specifications logical line by logical line until the *EOF control card is encountered and the loop is exited.

To process the specifications for a logical line the program first takes the two form cards, and stores columns 2-80 of the first and 1-53 of the second as a character string variable of length 132 in the format table. Next GIPSY scans across this string looking for slashes that indicate print positions. When one is found, its column number is placed in the array POSITION(k) (from the second group described above), the slash is replaced by a blank, the subscript incremented, and the scan then continues in this fashion until all 132 print positions have been examined.

THE FORMAT TABLES

Shown below is the internal representation of the format for Figure 5.

A. FORMAT TABLE:

<table>
<tr><th></th><th>FORM</th><th>CONTENT</th><th>NOCONTENT</th><th>SUPPRESS</th><th>SAMEPAGE</th><th>PAGEHEAD</th><th>OVERPRINT</th><th>TOPLINE</th><th>SKIPLINES</th><th>LASTLINE</th><th>INDENT</th><th>STOPCOL</th><th>TITLEPAGE</th></tr>
<tr><td>1</td><td>BIBLIOGRAPHY</td><td>(NULL STRING)</td><td>/</td><td></td><td>/</td><td>/</td><td>/</td><td>12</td><td>1</td><td>54</td><td>43</td><td>100</td><td></td></tr>
<tr><td>2</td><td>(132 BLANKS)</td><td>|./<|,// "|,"// |,// |,//@|@@>/</td><td></td><td>/</td><td></td><td></td><td></td><td>3</td><td>1</td><td>52</td><td>37</td><td>100</td><td></td></tr>
<tr><td>3</td><td>(132 BLANKS)</td><td>**|**/</td><td></td><td>/</td><td></td><td></td><td></td><td>0</td><td>1</td><td>54</td><td>41</td><td>100</td><td></td></tr>
<tr><td>4</td><td>(132 BLANKS)</td><td>|/</td><td></td><td></td><td></td><td></td><td>/</td><td>55</td><td>1</td><td>55</td><td>21</td><td>100</td><td></td></tr>
<tr><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

C. MISC. VECTORS:

	POSITION
1	25
2	30
3	39
4	49
...	

	VARIABLE
1	98(COUNTER)
2	1(AUTHOR)
3	2(TITLE
4	3(PUBLICATION)
5	4(PUBLISHER)
6	5(DATE)
7	11(DESCRIPTORS)
8	97(PAGENO)
...	

	VARNO
1	6(PAGES)

B. CONDITIONAL TABLE:

<table>
<tr><th></th><th>INSERT</th><th>CONDITION</th><th>NOVAR</th></tr>
<tr><td>1</td><td>(</td><td>-4</td><td>0</td></tr>
<tr><td>2</td><td>.</td><td>4</td><td>0</td></tr>
<tr><td>3</td><td>), |.</td><td>-4</td><td>1</td></tr>
<tr><td>...</td><td></td><td></td><td></td></tr>
</table>

Figure 9

Next GIPSY looks at the fields in columns 54 to 80 of the second form card and extracts the values and places them in the appropriate arrays in the format table. At this point the field in column 55 (NOCONTENT) is checked for a slash, if one is present the remainder of this section is bypassed, and processing of the next logical line is begun. If, however, a content card is present the PACKHOLD procedure is used to read and pack all the content cards into HOLD.

Throughout the rest of this section HOLD is constantly being picked apart, the information is processed and placed in arrays. Whenever anything is extracted it is replaced by a symbol so that when processing is completed what remains is a symbolic representation of the content card. This string is then placed in the format table along with all the information it represents.

As HOLD passes through this preprocessing section of coding the first items extracted are all the conditional expressions in that logical line and each is replaced in HOLD by the character '@'. Each conditional expression is decomposed and its pieces placed in a table where the subscript corresponds to the number of the occurrence of conditional expressions since the beginning of the format specifications.

The data to the left of the slash is further decomposed in that the variables are replaced by the '|' character. These variables' numbers are placed in an array VARNO(i) and the number of variables in the expression is placed in the array NOVAR(i). The remaining character string is placed in the array INSERT(i). The variable to the right of the slash is looked up, and its number, prefixed with the sign that was already there, is placed in the array CONDITION(i).

After all the conditional expressions have been processed, GIPSY scans HOLD looking for variables and whenever one is encountered it is replaced in HOLD by the character '|' and its number is placed in the table in array VARIABLE(i).

As the last step in preprocessing, the logical line HOLD is scanned looking for fields that have been concatenated (indicated by double slashes) to form a single new field. The program then uses the angular brackets to delimit this new field. Once HOLD has been completely processed, its contents are placed in the format table in the array CONTENT(i), and the program loops back to commence processing the next logical line.

D. THE SCREENING EXPRESSION

The section of the program that processes the *SCREEN control card is without a question the most interesting. Here too, the "preprocessing" philosophy prevailed. This section also makes use of a table. The screen table is made up of 5 one-dimensional arrays, each corresponding to a column of the table. The rows in the table correspond to each 'character string', and in the same sequence that they occur. Frequent referral to Figure 10 will aid in making the following paragraphs clearer. The first step is to use the PACKHOLD procedure to obtain the boolean expression that describes the screening criteria. HOLD is processed from left to right picking off the symbols one at a time.

1. The Screen Table

As each symbol is picked off it is examined for certain characteristics which indicate to the program how it is to be processed. If the symbol is a blank then it is discarded, if it is a right or left parenthesis it is left as is, and the next symbol is extracted. The key to the procedure is a symbol enclosed in single quotes, this is a literal character string that is going to be looked for later in the data base. When one is encountered, the quotes are stripped off and it is placed in the screen table in the array SEEK(i), it is replaced in HOLD by a slash, and then i, the row number in the table, is increased by one. Consequently, the rows in the screen table correspond to each of these character strings in the order that they are encountered.

If the symbol is one of the eight relational operators it is removed from HOLD and placed in the table in the array LOCATOR(i), the program then looks backward in LOCATOR and if any of the previous positions are blank they are filled with this same operator. Similarly, if the

PROCESSING A SCREEN

a. ORIGINAL EXPRESSION:

HOLD = ('SIM PROG' OR 'LIST PROC') AND 'LANG COMP' IN
    DESCRIPTORS AND ('1964' OR '1965' OR '1966' IN DATE)
    AND NOT ('GEH' OR 'YCC') AT LOCATION

b. ABBREVIATED FORM:

HOLD = (/0/)A/A(/0/0/)AN(/0/)

c. COMPLETELY PARENTHESIZED FORM:

SEEKER = (((((/0/)A/)A(((/0/)0/)))A((/0/)N)))

SCREEN TABLE

	SEEK	LOCATOR	ENTRY	TRU	FALS
1	SIM PROG	IN	11	3	2
2	LIST PROC	IN	11	3	-2
3	LANG COMP	IN	11	4	-2
4	1964	IN	5	7	5
5	1965	IN	5	7	6
6	1966	IN	5	7	-2
7	GEH	AT	10	-2	8
8	YCC	AT	10	-2	-1

Figure 10

symbol is an entry name, it is removed from HOLD, looked up in the entry name table, and its entry number is placed in the screen table in the array ENTRY(i), the program then works backward in ENTRY placing this number in all unfilled positions. At this point it can be seen that to each row in the screen table there corresponds a slash in HOLD, and this slash (and each row in the table) represents a simple boolean expression as defined in Section VI-A. Finally, if the symbol is a boolean operator AND, OR or NOT, all characters but the first one of the operator name are removed from HOLD.

After HOLD has been processed in this manner and is in abbreviated form, the expression is completely parenthesized according to the normal hierarchy of boolean operations. To do this requires three passes through HOLD from left to right. On the first pass each N (NOT) operator and its associated operand are enclosed in parentheses. Since NOT is a unary operator, the operator and the operand are exchanged in position to provide consistancy for a routine that is used later. On the second pass the A (AND) operator and both of its operands are enclosed in parentheses. Then, O (OR) operators and both their operands are parenthesized on the last pass. Throughout these three passes, previous parenthesization is taken into account in determining the operands which are to be parenthesized at the current step. All this done, the expression is then itself enclosed in parentheses and stored in a variable named SEEKER.

2. Expression "Pre-Evaluation"

The final phase in the processing of a screening expression utilizes a few shortcuts from Boolean algebra to "pre-evaluate" the boolean expression. The algorithm which does this starts with the first character in SEEKER and moves to the right one character at a time until SEEKER has been completely processed.

As the algorithm moves across SEEKER looking at each character, it keeps track of the parenthesis level with the variable M. When it encounters a left paren, M is incremented by one and when it encounters a right paren, M is decremented by one. If in processing SEEKER the character being looked at is a boolean operator, it is simply ignored and the next character is examined.

When a slash (a simple boolean expression) is encountered the procedure NEXTLINE is called twice, once with the value "false" supplied and once with the value "true" supplied. Roughly speaking NEXTLINE looks at the remainder of the screening criteria and says that if in the evaluation of the screen this simple boolean expression is true (or false) then the next simple boolean expression that must be evaluated is "i", the row number in the screening table.

NEXTLINE's operation is centered around the fact that in the logical expression (X OR Y), if the value of X is true then there is no need to evaluate Y, since its value has no effect on the overall value of the expression.

Only if X is false must Y be evaluated. The same is true in the expression (X AND Y) if the value of X if false. Only if X is true must Y be evaluated.

NEXTLINE records the current parenthesis level and proceeds across SEEKER adjusting the parenthesis level until it comes to the boolean operator that is associated with the slash that initiated the procedure call. Depending on which operator it is and also on the value supplied, the procedure then locates the next simple boolean expression that must be evaluated. This expression's row number in the screen table is coded and then returned.

The first call to NEXTLINE supplies the value "false," the coded number that NEXTLINE returns is then placed in the screen table in the array FALS(i). The second time NEXTLINE is called the value "true" is supplied and the value returned is placed in the table in the array TRU(i). The reason that the value returned by NEXTLINE is coded is that for each call there are three possible evaluation results: (1) there are no more simple boolean expressions to be evaluated and the final value of the screen is "true," (2) there are no more simple boolean expressions to be evaluated and final value of the screen is "false," and (3) row "i" in the screen table must be processed next.

The end result of this preprocessing of the screening criteria is that later in the program when a data record is being screened for inclusion in the output, the program enters the screen table at row one and evaluates this first simple boolean expression. Depending on the outcome of this evaluation it looks in the appropriate column, either TRU(i) or FALS(i) in the screen table to see what to do next. If the number found there is positive, it enters the table again at this indicated row. If the number is negative, then it knows that no more evaluations are necessary and that the final value of the screen expression is either true or false depending on whether the number there is -1 or -2.

E. THE REORDERING PROCEDURE

When the *SETUP control card is processed the array ORDER(i) is initialized by a DO LOOP such that ORDER(i) = i. In the section of the program that screens and processes the data records for output, the data records are accessed by row number in TABLE(i,j) through the contents of ORDER, i.e., TABLE(ORDER(i),j).

The first step in processing the *ORDER control card is to pick off the entry name and look its number up in the list of entry names. This number then becomes the column number, j, in TABLE(i,j).

The reordering process discussed in Section VII-B then becomes a matter of rearranging the contents of ORDER. The algorithm which does this sorting is efficient, in that in the worst case, the original order completely reversed, it only requires N passes through the list, where N is the number of items in the list.

The algorithm wanders through the proper column of TABLE fetching the entries by row number based on the contents of ORDER. It compares the first two members of the list, if the first member is larger than the second then their positions are exchanged so that member 1 is now member 2 and vise versa. If they are already in proper order nothing is done. Next members 2 and 3 are compared and if member 2 is larger than member 3 then their positions are exchanged. If they are already in the proper order nothing is done and the next two members are compared. This process is continued until the last two members of the list have been compared.

At this point a flag is tested to see if there were no exchanges made during this pass through the list. If such is the case, the list is in proper order and the procedure halts. If an exchange was made, the procedure loops back through the list and the process is repeated.

F. OUTPUT PROCESSING

Certainly the most complex section of GIPSY is the one that screens and outputs the data records. When this section of coding is entered the *PRINT control card is checked for the presence of the character string 'SCREEN'. If it is found, a flag is set to indicate to the program that the data records are to be compared against the most recent screening criteria and only those which satisfy the criteria are to be printed.

A few variables are then initialized, one of which is Q (initialized to one). Q is used as a row marker in the format table. The program enters the format table at row Q and checks TITLEPAGE(Q) for the presence of a slash. If a slash is found, the program goes into a loop which prints FORM(Q) according to the dictates of the rest of line Q in the table and then increments Q by one. The loop is exited the first time that TITLEPAGE(Q) equals a blank.

At this point another row marker for the format table, V, is set equal to Q. V serves two purposes; it acts as a flag which indicates the presence of page headings in the format specifications, and it marks the beginning line number in the format table of these page headings. If PAGEHEAD(Q) contains a slash then the program goes into a loop which prints FORM(Q) according to the dictates of the rest of line Q in the format table, and then Q is incremented by one. The loop is exited the first time that PAGEHEAD(Q) equals a blank.

Q is now the number of the first line in the format table which represents the first logical line that is to be processed for each data record selected. Rows V to Q in the format table are the format specifications to be processed each time that a new page is ejected.

With these administrative details out of the way GIPSY is ready to retrieve and process the data records. The remainder of this section in the program is contained in a loop that is traversed once for each record in the data file. It is here that the variable ORDER(i) is used to retrieve the data records. The contents of ORDER(i) are accessed sequentially beginning with i equal to one.

1. Screening the Records

The program looks at the screen flag, if it is not set then the next group of instructions are skipped and all records are processed and printed. If the screen flag is set GIPSY starts with row one in the screen table and fetches SEEK(1), LOCATOR(1), and ENTRY(1). Then the entry pointed to by TABLE(ORDER(1), ENTRY(1)) is fetched, and compared with SEEK(1) according to the dictates of LOCATOR(1). If the result of this comparison is "true" then TRU(1) is stored in M, if the result is false then FALS(1) is stored in M.

The program then examines M, if it is greater than zero then further screening is required and the entry pointed to by TABLE(ORDER(1), ENTRY (M)) is fetched and compared with SEEK(M) according to the dictates of LOCATOR(M), and so on until the value in M is negative. This indicates that no further screening is required. If M is a -1 then the record satisfied the screening criteria and is passed on to be printed. If M is a -2 then the record failed to satisfy the screening criteria and the rest of this section is bypassed, and the next record is obtained.

2. Formatting the Output

A data record reaches this point in output processing in one of two ways, it satisfactorily passed the screen or it was not required to be screened. The variable COUNTER is incremented by one and a loop is entered that starts at row Q in the format table, and utilizing the current data record, cycles through the rest of the table printing the specified information. The first step in the loop is to check the logical line to see if there is a slash in NOCONTENT(i). If one is present the program skips the next group of instructions which are concerned with processing the content of the line. Otherwise CONTENT(i) is fetched from the format table and is processed.

In processing a logical line for printing, the contents of FORM(i) are saved in a temporary variable for over printing purposes, and CONTENT(i) is processed character by character. HOLD, a character variable of length 3000, is used to hold the logical line as it is being assembled for printing.

First the character string from position 1 to POSITION(k)-1 in FORM(i) is placed in HOLD. This character string contains any associated text that was on the format cards. Next the first field from CONTENT(i) is processed, placing literals, appropriate data record entries or variables, and/or conditional information in HOLD, beginning in position POSITION(k). Then the next piece of FORM(i) is placed in HOLD and the next field processed and placed in HOLD, and so on until the whole logical line, fully processed, is in HOLD.

In processing the individual field from CONTENT(i) the possibility of the field being a composite of fields and the possibility of conditional expressions being included, must be taken into account. If a conditional expression is present then condition CONDITION(j) must be checked. If the condition is not satisfied then the expression is ignored and processing of CONTENT(i) is resumed. If the condition is satisfied then INSERT(j) is processed as if it were a subfield. If the field is a composite of fields then all of the individual fields are processed as subfields. All the subfields processed, they are then placed in HOLD as if they were one field.

If in processing any field or subfield, an entry to be inserted turns out to be the null entry, the program checks SUPPRESS(i) for the presence of a slash. If it is blank the program simply treats the null string as a valid entry and continues processing in the normal manner. If, however, it is a slash then the program eliminates all literals associated with that entry. If the whole field then turns out to be null, all associated text is also eliminated. This results in the null string being placed in HOLD for the corresponding print position, POSITION(k).

3. Printing the Output

The program is now ready to begin printing. HOLD contains the logical line, and FORM(i) contains the text to be overprinted. The logical line must be checked to see if it is longer than a physical line, in other words the length of HOLD is checked against STOPCOL(i). If it is longer, then the program takes HOLD at the STOPCOL(i) position and works backward until it comes to the first blank. The first physical line is then taken up to that point. This character string is padded on the right with blanks to fill out to 132 characters and the result is placed in an array named OUTLINE(n). A loop is then entered that takes the next available OUTLINE(n) and fills in blanks from position 1 to position IDENT(i)-1. The program takes the next STOPCOL(i) - IDENT(i) characters of HOLD, finds the end of the last word, pads on the right with blanks, places the result in the current OUTLINE(n), and then increments N. The loop is exited when the logical line is in a form suitable for printing.

The program now looks to see at what point on the page the last logical line completed printing. It compares this with the number of lines that are required to print the current logical line and if this sum is greater that the value of LASTLINE(i), the page number is printed (if specified) and a new page is ejected.

At this point, the number of lines indicated by TOPLINE(i) are skipped and OVERPRINT(i) is checked for a slash. If a slash is present then FORM(i) is printed on the same line COLOR-1 times and then OUTLINE(1) is printed on this same line. If a slash is not present then just

OUTLINE(1) is printed. If required, the program enters a loop that skips the number of lines indicated by the value of SKIPLINES(i) and prints OUTLINE(i) and then increments n. The loop is exited when the logical line is completely printed.

This point in the program marks the end of two loops, the loop that is processing the logical lines and the loop that is processing the data records. When the last logical line associated with the last data record has been processed, control is transferred to the section of the program that is to process the next control card.

G. ELAPSED TIME PROCESSING

When the *TIME control card is encountered the current time of day is obtained immediately. Next, the text to be printed and its attendent formatting information is obtained. The time of day is returned as a character string, consequently, the various substrings that represent hours, minutes, etc., must be converted to fixed binary numbers in order that arithmetic operations may be performed. All these quantities are converted to milliseconds and summed. From this quantity is subtracted the value of CLOCK (described in Section VIII_A) and the result converted to seconds. CLOCK is updated so that it now contains the time of day, in milliseconds, of the most recent processing of the *TIME control card. Finally, the elapsed time, in seconds, followed by the text is printed according to the format specified.

IX. CONCLUDING REMARKS

There are several modifications and extensions that could be made to the program which would extend the scope of applicability and the usefulness of GIPSY. The first such modification would be to adapt it to operate in a time sharing mode instead of its present batch mode. An efficient algorithm could be included that would provide for swapping WORD(i) vectors between core and a disk, such that only one vector be in core at any one time. This would reduce the core storage requirements of GIPSY. Also, it would be relatively easy to add to GIPSY the capability of adding, deleting, and/or modifying the data file. A natural language preprocessor that would make it easier to specify the different tasks that are to be performed could be added, too. As more experience with using GIPSY is accumulated, other improvements will probably suggest themselves, also.

PROGRAM LISTING

```
/*    GENERAL INFORMATION PROCESSING SYSTEM      */

  GIPSY: PROCEDURE OPTIONS(MAIN);
         ON ENDFILE(SYSIN) GOTO COMPLETED;

/*    DECLARATION OF STATIC VARIABLES    */

       DECLARE (A,B,C,D,E,G,H,I,J,K,L,M,N,P,Q,R,S,T,U,V,W,X,Y,GG,HH,
               CARDS,COLUMN,COLUMNS,ENTITY,ENTITY2,ESTIMATE,LINES,KK,
               MAXFORM,NC_RCRDS,PAGENO,PAGES,SCREENFLAG,JJ) BIN FIXED,
               (LL,MM,IDCOL,COLS,NOLINES,COLOR,SIZE) BIN FIXED,
               BIG BIN FIXED INIT(32767),
               COUNTER BIN FIXED INIT(0),
               F BIN FIXED INIT(0),
               (CLOCK,STORAGE) BIN FIXED(31),
               (CONDITION,POSITION,VARIABLE,VARNO,
               NOVAR)(100) BIN FIXED,
               (ENTRY,FALS,TRU)(50) BIN FIXED;

       DECLARE BIGBLANK CHAR(132) INIT((132) ' '),
               BLANK CHAR(1) INIT(' '),
               BLANKS CHAR(4) INIT('    '),
               CARD CHAR(80),
               (HOLD,HOLD3,HOLD4,HOLD5,HOLD6) CHAR(3000) VARYING,
               (HOLD1,HOLD2) CHAR(200) VARYING,
               INSERT(100) CHAR(25) VARYING,
               LOCATOR(50) CHAR(2),
               NIL CHAR(2) VARYING INIT(''),
               NUMBER CHAR(9) VARYING,
               OUTLINE(50) CHAR(132),
               SEEK(50) CHAR(25) VARYING,
               SEEKER CHAR(300) VARYING,
               SLASH CHAR(1) INIT('/'),
               TEMP CHAR(80) VARYING,
               TIMES CHAR(9),
               Z CHAR(1),
               ZZ CHAR(2),
               CONTROL(11) CHAR(7) INIT('DATA   ','CHANGE ',
               'PRINT  ','FORMAT ','SCREEN ','TIME   ','HEADING',
               'PRINTCO','ENDJOB ','ORDER  ','SETUP  ');

/*    CALCULATIONS TO DETERMINE VARIABLE DIMENSIONS  */

           TIMES = TIME;
           HH = SUBSTR(TIMES,1,2);
           JJ = SUBSTR(TIMES,3,2);
           KK = SUBSTR(TIMES,5,2);
           LL = SUBSTR(TIMES,7);
           CLOCK = LL + 1000*KK + 60000*JJ + 3600000*HH;
           GET EDIT (CARD) (COLUMN(1),A(80));
           IF SUBSTR(CARD,2,4) = 'LIST' THEN DO I = 1 TO BIG;
                PUT PAGE LINE(13);
                DO W = 1 TO 46;
                GET EDIT (CARD) (COLUMN(1),A(80));
                PUT SKIP EDIT (CARD) (COLUMN(26),A(80));
                END;
                END;
           SETUPS: CARDS = 463;
                   ESTIMATE = 100;
                   ENTITY = 10;
                   NOLINES = 0;
                   COLS = 70;
                   COLOR,SIZE = 3;
                   IDCOL = 77;
                   GET DATA ;
                   STORAGE = CARDS * COLS;
                   IF NOLINES = 0 THEN NOLINES = ENTITY;
                   NN = 80 - IDCOL + 1;
                   PUT SKIP EDIT
                         (STORAGE,' BYTES OF CORE HAVE BEEN RESERVED')
                         (COLUMN(8),F(10),A);
                   ENTITY2 = ENTITY + 2;
                   L = STORAGE / BIG + 1;
                   CALL START;


            START: PROCEDURE;
               ON ENDFILE(SYSIN) GOTO COMPLETED;
```

```
/*    DECLARATION OF DYNAMIC VARIABLES    */

                DECLARE ATTRIBUTES(ENTITY) CHAR(50) VARYING,
                        CONTENT(NOLINES) CHAR(100) VARYING,
                        FORM(NOLINES) CHAR(132) VARYING,
                        (INDENT,LASTLINE,SKIPLINES,STOPCOL,
                        TOPLINE)(NOLINES) BIN FIXED,
                        (NOCONTENT,OVERPRINT,PAGEHEAD,SAMEPAGE,
                        SUPPRESS,TITLEPAGE)(NOLINES) CHAR(1),
                        ORDER(ESTIMATE) BIN FIXED,
                        SERIAL CHAR(MM),
                        SERIALS(ESTIMATE) CHAR(MM),
                        TABLE(ESTIMATE,ENTITY2) BIN FIXED,
                        WORD(L) CHAR(32767) VARYING,
                        ENTER(8) LABEL,
                        SEGMENT(12) LABEL;


/*    VARIABLE INITIALIZATION    */

                DO I = 1 TO ESTIMATE;
                     ORDER(I) = I;
                     END;
                DO I = 1 TO L;
                     WORD(I) = NIL;
                     END;
                DO I = 1 TO ENTITY;
                     ATTRIBUTES(I) = NIL;
                     END;
                B = 0;
                COUNTER = 0;


/*    PROCESS SETUP CONTROL CARD    */

                GET EDIT (CARD) (COLUMN(1),A(80));
                CALL PACKHOLD;
                DO WHILE (COLUMN ¬> J);
                     C = INDEX(SUBSTR(HOLD,COLUMN),SLASH);
                     B = B + 1;
                     ATTRIBUTES(B)=SUBSTR(HOLD,COLUMN,(C-1));
                     COLUMN = COLUMN + C;
                     END;
                CALL GOTOA;
                GOTO SEGMENT(A);
```

```
/*    DATA STORAGE SECTION    */

       SEGMENT(1): GET EDIT (CARD) (COLUMN(1),A(80));
         ENTER(1): K,L,M,N = 1;
                   DO WHILE (SUBSTR(CARD,2,3) ¬= 'EOF');
                        CALL PACKHOLD;
                        IF (M + J - 1) > BIG THEN DO:
                             L = L + 1;
                             M = 1;
                             END;
                        DO WHILE (COLUMN ¬> J);
                             C=INDEX(SUBSTR(HOLD,COLUMN),SLASH);
                             HOLD5 = SUBSTR(HOLD,COLUMN,C-1);
                             WORD(L) = WORD(L)||HOLD5;
                             TABLE(K,ENTITY2) = L;
                             TABLE(K,N) = M;
                             SERIALS(K) = SERIAL;
                             M = M + C - 1;
                             N = N + 1;
                             COLUMN = COLUMN + C;
                             END;
                        DO I = N TO ENTITY + 1;
                             TABLE(K,I) = M;
                             END;
                        NO_RCRDS = K;
                        K = K + 1;
                        N = 1;
                        END;
                   GOTO SEGMENT(12);

/*    PROCESS CHANGE CONTROL CARD    */

       SEGMENT(2): GET DATA;
         ENTER(2): GOTO SEGMENT(12);

/*    PROCESS PRINT CONTROL CARD    */

       SEGMENT(3): C=INDEX(CARD,'SCREEN');
                   IF C=0 THEN SCREENFLAG=0; ELSE SCREENFLAG=1;
                   U,V,PAGENO = 0;
                   Q = 1;
                   DO W=1 TO 50;
                        OUTLINE(W)=BIGBLANK;
```

```
          END;
IF TITLEPAGE(Q) = SLASH THEN DO;
DO WHILE(TITLEPAGE(Q) = SLASH);
          IF SAMEPAGE(Q)=SLASH | (F+TOPLINE(Q)+1) >
                    LASTLINE(Q) THEN DO;
                    PUT PAGE;
                    F=0;
                    END;
          M=Q;
          CUTLINE(49) = FORM(M);
          CALL CPRINT;
          Q=Q+1;
          END;
          PUT PAGE;
          F = 0;
          END;
V=Q;
IF PAGEHEAD(Q) = SLASH THEN DO;
          PUT PAGE;
          F = 0;
          DO WHILE (PAGEHEAD(Q) = SLASH);
                    M = Q;
                    CUTLINE(49) = FORM(M);
                    CALL CPRINT;
                    Q = Q + 1;
                    END;
          END;
IF SAMEPAGE(Q) ¬= SLASH THEN PAGENO = PAGENO + 1;
DO L=1 TO NO_RCRDS;
          K=ORDER(L);
          G,H,GG,HH = 1;
          IF SCREENFLAG = 1 THEN DO;
                    M = 1;
      OUT13:        TEMP = SEEK(M);
                    N = ENTRY(M);
                    ZZ = LOCATOR(M);
                    LL = TABLE(K,ENTITY2);
                    KK = TABLE(K,N);
                    JJ = TABLE(K,N+1);
                    HOLD = SUBSTR(WORD(LL),KK,JJ-KK);
                    IF ZZ = 'IN' THEN DO;
                              C = INDEX(HOLD,TEMP);
                              IF C = 0 THEN GOTO OUT12;
                                        ELSE GOTO OUT11;
                              END;
                    IF ZZ = 'AT' THEN DO;
```

```
            IF TEMP =SUBSTR(HOLD,1,LENGTH(TEMP))
                 THEN GOTO OUT11:
                 ELSE GOTO OUT12:
            END;
       IF ZZ = 'GT' THEN DO;
            IF TEMP > HOLD THEN GOTO OUT11:
                 ELSE GOTO OUT12;
            END;
       IF ZZ = 'LT' THEN DO;
            IF TEMP < HOLD THEN GOTO OUT11:
                 ELSE GOTO OUT12;
            END;
       IF ZZ = 'GE' THEN DO;
            IF TEMP ¬< HOLD THEN GOTO OUT11:
                 ELSE GOTO OUT12;
            END;
       IF ZZ = 'LE' THEN DO;
            IF TEMP ¬> HOLD THEN GOTO OUT11:
                 ELSE GOTO OUT12;
            END;
       IF ZZ = 'EQ' THEN DO;
            IF TEMP = HOLD THEN GOTO OUT11:
                 ELSE GOTO OUT12;
            END;
       IF ZZ = 'NE' THEN DO;
            IF TEMP ¬= HOLD THEN GOTO OUT11:
                 ELSE GOTO OUT12;
            END;
       PUT SKIP EDIT
            ('ILLEGAL OPERATOR ON SCREEN CARD ='
            ,' ',TEMP) (COLUMN(8),A,A,A(2));
       STOP;
OUT11: M = TRU(M);
       IF M > 0 THEN GOTO OUT13;
       IF M = -1 THEN GOTO OUT14;
            ELSE GOTO OUT22;
OUT12: M = FALS(M);
       IF M > 0 THEN GOTO OUT13;
       IF M = -1 THEN GOTO OUT14;
            ELSE GOTO OUT22;
       END:
  OUT14: COUNTER = COUNTER + 1:
  DO I=0 TO MAXFORM;
       OUTLINE(50) = FORM(I);
       S,D,P = 1;
       IF NOCONTENT(I)=SLASH THEN DO;
```

```
     OUTLINE(D) = OUTLINE(50);
     GOTO OUT8;
     END;
HOLD1 = CONTENT(I);
T=LENGTH(HOLD1);
DO WHILE (P¬>T);
     J=POSITION(H)-S;
     HOLD2=SUBSTR(OUTLINE(50),S,J);
     IF SUBSTR(HOLD1,P,1)='<' THEN DO;
          C=INDEX(SUBSTR(HOLD1,P),'>');
          HOLD3=SUBSTR(HOLD1,P+1,C-2);
          P=P+C+1;
          END;
          ELSE DO;
          C=INDEX(SUBSTR(HOLD1,P),SLASH);
          HOLD3=SUBSTR(HOLD1,P,C-1);
          P=P+C;
          END;
     HOLD4=NIL;
     N = 1;
     Y=LENGTH(HOLD3);
     DO WHILE (N¬>Y);
          C=INDEX(SUBSTR(HOLD3,N),SLASH);
          IF C=0 THEN TEMP=SUBSTR(HOLD3,
          N);
               ELSE TEMP=SUBSTR(HOLD3,N,
               C-1);
          E=LENGTH(TEMP);
          N=N+E+2;
          HOLD=NIL;
          R=1;
          DO WHILE (R¬>E);
               IF SUBSTR(TEMP,R,1) = '@'
                    THEN DO;
                    C = CONDITION(GG);
                    C = ABS(C);
                    M=TABLE(K,C+1) -
                    TABLE(K,C);
                    C=CONDITION(GG);
                    IF (C>0 & M¬=0) |
                    (C<0 & M=0) THEN DO;
                    HOLD5 = NIL;
                    C = 1;
                    M=LENGTH(INSERT(GG));
                    DO WHILE (C¬>M);
                    Z=SUBSTR(INSERT(GG),
          C,1);
    IF Z='|' THEN DO;
    X=VARNO(HH);
    HH = HH+1;
    IF X=97 THEN DO;
    NUMBER = PAGENO;
    DO WHILE (SUBSTR(
          NUMBER,1,1) =
          BLANK);
    NUMBER=SUBSTR(NUMBER,
    2); END;
    HOLD5=HOLD5||NUMBER;
    GOTO OUT32;
    END;
    IF X=98 THEN DO;
    NUMBER = COUNTER;
    NUMBER=SUBSTR(NUMBER,
          9-SIZE+1,SIZE);
    HOLD5=HOLD5||NUMBER;
    GOTO OUT32;
    END;
    IF X=99 THEN DO;
    HOLD5=HOLD5||
          SERIALS(K);
    GOTO OUT32;
    END;
    LL=TABLE(K,ENTITY2);
    JJ=TABLE(K,X);
    KK=TABLE(K,X+1);
    HOLD6=SUBSTR(WORD(LL)
    ,JJ,KK-JJ);
    HOLD5=HOLD5||HOLD6;
    END;
    ELSE HOLD5=HOLD5||Z;
    OUT32: C=C+1;
    END;
    END;
    ELSE DO;
    HOLD5 = NIL;
    HH=HH+NOVAR(GG);
    END;
    GG = GG + 1;
    GOTO OUT27;
    END;
IF SUBSTR(TEMP,R,1)='|'
THEN DO;
```

```
               X=VARIABLE(G);
               G = G + 1;
               IF X=97 THEN DO;
                    U=I;
                    COLUMN=H;
                    GOTO OUT15;
                    END;
               IF X = 98 THEN DO;
                    NUMBER=COUNTER;
                    NUMBER=SUBSTR(
                    NUMBER,9-SIZE+1,
                    SIZE);
                    HOLD5=NUMBER;
                    GOTO OUT19;
                    END;
               IF X = 99 THEN DO;
                    HOLD5=SERIALS(K)
                    ;
                    GOTO OUT19;
                    END;
               LL=TABLE(K,ENTITY2);
               JJ=TABLE(K,X);
               KK=TABLE(K,X+1);
               HOLD5=SUBSTR(WORD(LL)
               ,JJ,KK-JJ);
               OUT19: IF LENGTH
               (HOLD5)=0&SUPPRESS(I)
               =SLASH THEN DO;
                    HOLD = NIL;
                    GOTO OUT6;
                    END;
               END;
               ELSE HOLD5 =
               SUBSTR(TEMP,R,1);
          OUT27: HOLD=HOLD||HOLD5;
          R = R + 1;
          END;
     OUT6: HOLD4 = HOLD4||HOLD;
     END;
R = LENGTH(HOLD4);
IF R=0 & SUPPRESS(I)=SLASH THEN DO;
     W = POSITION(H) - 1;
     SUBSTR(OUTLINE(50),S,W) =
     SUBSTR(BIGBLANK,S,W);
     HOLD2=SUBSTR(BIGBLANK,1,
          LENGTH(HOLD2));
```

```
                 HOLD4 = BLANK;
                 END;
          H = H + 1;
          IF (R+J+S-1) > STOPCOL(I) THEN
                 GOTO OUT7;
          SUBSTR(OUTLINE(D),S,R+J)=
                 HOLD2||HOLD4;
          S = S + J + R;
          END;
GOTO OUT8;
OUT7: N = STOPCOL(I) - S -J;
DO WHILE (SUBSTR(HOLD4,N,1)¬=BLANK);
          N = N - 1;
          END;
N = N - 1;
SUBSTR(OUTLINE(D),S,N+J) = HOLD2||
          SUBSTR(HOLD4,1,N);
HOLD4 = SUBSTR(HOLD4,N+2);
DO WHILE (LENGTH(HOLD4) > 0);
          D = D + SKIPLINES(I);
          W = INDENT(I);
          N = STOPCOL(I) - W + 1;
          IF LENGTH(HOLD4) > N THEN DO;
                 DO WHILE (SUBSTR(HOLD4,N,1)
                          ¬= BLANK);
                          N = N - 1;
                          END;
                 N = N - 1;
                 SUBSTR(OUTLINE(D),W,N)
                 = SUBSTR(HOLD4,1,N);
                 HOLD4 = SUBSTR(HOLD4,N+2);
                 END;
                 ELSE DO;
                 SUBSTR(OUTLINE(D),W,
                          LENGTH(HOLD4)) = HOLD4;
                 HOLD4 = NIL;
                 END;
          END;
OUT8: IF OUTLINE(1) = BIGBLANK &
OUTLINE(50) = BIGBLANK THEN DO;
          IF SAMEPAGE(I) = SLASH THEN DO;
                 IF M ¬= 0 THEN DO;
                          M = U;
                          OUTLINE(49)=FORM(M);
                          KK = TOPLINE(M);
                          TOPLINE(M)=TOPLINE(M)-F-1;
```

```
               NUMBER = PAGENO;
               DO WHILE (SUBSTR(NUMBER,
                    1,1) = BLANK);
                    NUMBER=SUBSTR(NUMBER
                    ,2);
                    END;
               W=POSITION(COLUMN);
               CARDS = LENGTH(NUMBER);
               SUBSTR(OUTLINE(49),W,
               CARDS) = NUMBER;
               CALL OPRINT;
               TOPLINE(M) = KK;
               END;
          PUT PAGE;
          F = 0;
          PAGENO = PAGENO + 1;
          END;
     GOTO OUT15;
     END;
IF SAMEPAGE(I) = SLASH |
     (F+TOPLINE(I)+D) > LASTLINE(I)
     THEN DO;
     IF U ¬= 0 THEN DO;
          M = U;
          OUTLINE(49) = FORM(M);
          KK = TOPLINE(M);
          TOPLINE(M) = TOPLINE(M) - F -1;
          NUMBER = PAGENO;
          DO WHILE (SUBSTR(NUMBER,1,1) =
               BLANK);
               NUMBER = SUBSTR(NUMBER,2);
               END;
          W = POSITION(COLUMN);
          CARDS = LENGTH(NUMBER);
          SUBSTR(OUTLINE(49),W,CARDS) =
               NUMBER;
          CALL OPRINT;
          TOPLINE(M) = KK;
          END;
     PUT PAGE;
     PAGENO = PAGENO + 1;
     F = 0;
     IF V ¬= Q THEN DO M = V TO Q - 1;
          OUTLINE(49) = FORM(M);
          CALL OPRINT;
          END;
```

```
                                        END;
                                    PUT SKIP (TOPLINE(I)+1);
                                    F = F + TOPLINE(I) + 1;
                                    IF OVERPRINT(I) = SLASH THEN DO W = 1 TO
                                        COLOR - 1;
                                        PUT SKIP(0) EDIT (OUTLINE(50))
                                            (COLUMN(1),A);
                                        END;
                                    PUT SKIP(0) EDIT (OUTLINE(1))
                                        (COLUMN(1),A);
                                    OUTLINE(1) = BIGBLANK;
                                    IF D ¬= 1 THEN DO W = 2 TO D;
                                        PUT SKIP(1) EDIT (OUTLINE(W))
                                            (COLUMN(1),A);
                                        OUTLINE(W) = BIGBLANK;
                                        F = F + 1;
                                        END;
                                    OUT15: END;
                                OUT22: END;
                            IF U ¬= 0 THEN DO;
                                M = U;
                                OUTLINE(49) = FORM(M);
                                KK = TOPLINE(M);
                                TOPLINE(M) = TOPLINE(M) - F - 1;
                                NUMBER = PAGENO;
                                DO WHILE (SUBSTR(NUMBER,1,1) = BLANK);
                                    NUMBER = SUBSTR(NUMBER,2);
                                    END;
                                W = POSITION(COLUMN);
                                CARDS = LENGTH(NUMBER);
                                SUBSTR(OUTLINE(49),W,CARDS) = NUMBER;
                                CALL OPRINT;
                                TOPLINE(M) = KK;
                                PUT PAGE;
                                F = 0;
                                END;
                            GOTO SEGMENT(12);


/*     PROCESS FORMATTING INFORMATION    */


      SEGMENT(4): H,I = 0;
                  G,GG,HH = 1;
                  DO W = 1 TO 100;
                      NOVAR(W) = 0;
```

```
                   INSERT(W) = NIL;
                   END;
               GET EDIT (CARD) (COLUMN(1),A(80));

/*    PICK OFF INFORMATION FROM A CARD     */

               DO WHILE (SUBSTR(CARD,2,3) ¬= 'EOF');
                   I = I + 1;
                   FORM(I),CONTENT(I) = NIL;
                   FORM(I) = SUBSTR(CARD,2);
                   GET EDIT (CARD) (COLUMN(1),A(80));
                   FORM(I) = FORM(I)||SUBSTR(CARD,1,53);
                   COLUMN = 1;
                   DO WHILE (COLUMN ¬> 132);
                       C=INDEX(SUBSTR(FORM(I),COLUMN),SLASH);
                       IF C = 0 THEN GOTO OUT20;
                       SUBSTR(FORM(I),COLUMN+C-1,1) = BLANK;
                       H = H + 1;
                       POSITION(H) = COLUMN + C - 1;
                       COLUMN = COLUMN + C;
                       END;
                   OUT20: NOCONTENT(I) = SUBSTR(CARD,55,1);
                   SUPPRESS(I) = SUBSTR(CARD,56,1);
                   SAMEPAGE(I) =SUBSTR(CARD,57,1);
                   PAGEHEAD(I) = SUBSTR(CARD,59,1);
                   OVERPRINT(I) =SUBSTR(CARD,61,1);
                   TOPLINE(I) = SUBSTR(CARD,63,2);
                   SKIPLINES(I) = SUBSTR(CARD,65,2);
                   LASTLINE(I) = SUBSTR(CARD,69,2);
                   INDENT(I) = SUBSTR(CARD,72,3);
                   STOPCOL(I) = SUBSTR(CARD,76,3);
                   TITLEPAGE(I) = SUBSTR(CARD,80,1);
                   GET EDIT (CARD) (COLUMN(1),A(80));
                   IF NOCONTENT(I) = SLASH THEN GOTO ENTER(4);

/*    PACK INFORMATION INTO CONTENT(I)    */

                   CALL PACKHOLD;
                   COLUMN = 1;
                   J = LENGTH(HOLD);
                   DO WHILE (COLUMN ¬> J);
                       C = INDEX(SUBSTR(HOLD,COLUMN),'<');
                       IF C = 0 THEN GOTO OUT23;
                       N = INDEX(SUBSTR(HOLD,COLUMN+C),'>');
```

```
IF N = 0 THEN DO;
    PUT SKIP LIST (HOLD);
    PUT SKIP EDIT
    ('UNBALANCED DELIMITERS IN ABOVE')
    (COLUMN(1),A);
    STOP;
    END;
TEMP=SUBSTR(HOLD,COLUMN+C-1,N+1);
HOLD=SUBSTR(HOLD,1,COLUMN+C-2)||'@'||
    SUBSTR(HOLD,COLUMN+C+N);
J = J - N;
COLUMN = COLUMN + C;
TEMP = SUBSTR(TEMP,2,N-1);
C = INDEX(TEMP,SLASH);
IF C = 0 THEN DO;
    PUT SKIP LIST (TEMP);
    PUT SKIP EDIT
    ('DELIMITER MISSING IN ABOVE')
    (COLUMN(1),A);
    STOP;
    END;
HOLD1 = SUBSTR(TEMP,1,C-1);
TEMP = SUBSTR(TEMP,C+1);
X = 1;
Y = LENGTH(HOLD1);
DO WHILE (X ¬> Y);
    HOLD2 = NIL ;
    Z = SUBSTR(HOLD1,X,1);
    IF Z = '''' THEN DO;
        C=INDEX(SUBSTR(HOLD1,X+1),'''')
            ;
        IF C = 0 THEN DO;
            PUT SKIP LIST (HOLD1);
            PUT SKIP EDIT
        ('DELIMITER MISSING IN ABOVE')
            (COLUMN(1),A);
            STOP;
            END;
        HOLD2=HOLD2||SUBSTR(HOLD1,X+1,
            C-1);
        X = X + C;
        GOTO OUT26;
        END;
    IF Z<'A' | Z>'Z' THEN DO;
        HOLD2 = Z;
        GOTO OUT26;
        END;
    HOLD2 = HOLD2||Z;
    X = X + 1;
    DO WHILE (X ¬> Y);
        Z=SUBSTR(HOLD1,X,1);
        IF Z<'A' THEN GOTO OUT24;
        HOLD2 = HOLD2||Z;
        X = X + 1;
        END;
    OUT24: DO W = 1 TO B;
        IF HOLD2=ATTRIBUTES(W) THEN DO;
            VARNO(HH) = W;
            GOTO OUT33;
            END;
        END;
    IF HOLD2='PAGENO' THEN DO;
        VARNO(HH)=97;
        GOTO OUT33;
        END;
    IF HOLD2='COUNTER' THEN DO;
        VARNO(HH)=98;
        GOTO OUT33;
        END;
    IF HOLD2='IDENT' THEN DO;
        VARNO(HH)=99;
        GOTO OUT33;
        END;
    GOTO OUT29;
    OUT33: HOLD2 = '|';
    NOVAR(GG)=NOVAR(GG) + 1;
    HH = HH + 1;
    OUT29: X = X - 1;
    OUT26: INSERT(GG)=INSERT(GG)||HOLD2;
    X = X + 1;
    END;
Z = SUBSTR(TEMP,1,1);
TEMP = SUBSTR(TEMP,2);
DO W = 1 TO B;
    IF TEMP=ATTRIBUTES(W) THEN DO;
        IF Z='-' THEN CONDITION(GG)=-W;
            ELSE CONDITION(GG) = W;
        GOTO OUT30;
        END;
    END;
PUT SKIP LIST (TEMP);
PUT SKIP EDIT
```

```
    ('THE ABOVE IS AN INVALID OPERAND')
    (COLUMN(1),A):
    STOP:
    OUT30: GG = GG + 1:
    END:
OUT23: COLUMN = 1:
DO WHILE(COLUMN ¬>J):
    C=INDEX(SUBSTR(HOLD,COLUMN),SLASH):
    IF C = 0 THEN GOTO OUT5:
    TEMP = SUBSTR(HOLD,COLUMN,C-1):
    X = 1:
    Y = C - 1:
    DO WHILE (X ¬> Y):
        HOLD2 = NIL:
        Z = SUBSTR(TEMP,X,1):
        IF Z = '''' THEN DO:
            N=INDEX(SUBSTR(TEMP,X+1),''''):
            IF N = 0 THEN DO:
                PUT SKIP LIST (TEMP):
                PUT SKIP EDIT
            ('DELIMITER MISSING IN ABOVE')
                (COLUMN(1),A):
                STOP:
                END:
            HOLD2=HOLD2||SUBSTR(TEMP,X+1,
                C-1):
            X = X + C:
            GOTO OUT2:
            END:
        IF Z < 'A' | Z > 'Z' THEN DO:
            HOLD2 = Z:
            GOTO OUT2:
            END:
        HOLD2 = HOLD2||Z:
        X = X + 1:
        DO WHILE (X ¬> Y):
            Z=SUBSTR(TEMP,X,1):
            IF Z<'A' THEN GOTO OUT31:
            HOLD2 = HOLD2||Z:
            X = X + 1:
            END:
        OUT31: DO W = 1 TO B:
            IF HOLD2=ATTRIBUTES(W) THEN DO:
                VARIABLE(G) = W:
                GOTO OUT25:
                END:
```

```
                                        END;
                                  IF HOLD2 = 'PAGEND' THEN DO;
                                        VARIABLE(G) = 97;
                                        GOTO OUT25;
                                        END;
                                  IF HOLD2 = 'COUNTER' THEN DO;
                                        VARIABLE(G) = 98;
                                        GOTO OUT25;
                                        END;
                                  IF HOLD2 = 'IDENT' THEN DO;
                                        VARIABLE(G) = 99;
                                        GOTO OUT25;
                                        END;
                                  GOTO OUT35;
                                  OUT25: HOLD2 = '|';
                                  G = G + 1;
                                  OUT35: X = X - 1;
                                  OUT2: CONTENT(I)=CONTENT(I)||HOLD2;
                                  X = X + 1;
                                  END;
                             CONTENT(I) = CONTENT(I)||SLASH;
                             IF COLUMN + C > J THEN GOTO OUT5;
                             IF SUBSTR(HOLD,COLUMN+C,1) = SLASH
                                  THEN DO;
                                  CONTENT(I) = CONTENT(I)||SLASH;
                                  COLUMN = COLUMN + C + 1;
                                  END;
                                  ELSE COLUMN = COLUMN + C;
                             END;


/*    PARENTHESIZE CONCATENATED ENTRIES    */

                  OUT5: HOLD2 = CONTENT(I);
                        M = 1;
                        Q = LENGTH(HOLD2);
                        DO WHILE (M < Q);
                             TEMP = SUBSTR(HOLD2,M,2);
                             IF TEMP = SLASH||SLASH THEN DO;
                                  P = M + 2;
                            OUT9: Z = SUBSTR(HOLD2,P,1);
                                  DO WHILE (Z ¬= SLASH);
                                       P = P + 1;
                                       Z = SUBSTR(HOLD2,P,1);
                                       END;
                                  IF (P+1)¬> Q & SUBSTR(HOLD2,P+1,1)
```

```
                                = SLASH THEN DO:
                                P = P + 2;
                                GOTO OUT9:
                                END;
                            HOLD1 = SUBSTR(HOLD2,1,P-1)||'>':
                            HOLD1 = HOLD1||SUBSTR(HOLD2,P):
                            N = M - 1;
                            Z = SUBSTR(HOLD1,N,1):
                            DO WHILE (Z ¬= SLASH);
                                N = N - 1;
                                IF N = 0 THEN GOTO OUT28:
                                Z = SUBSTR(HOLD1,N,1);
                                END;
                            OUT28: HOLD2=SUBSTR(HOLD1,1,N)||'<';
                            HOLD2 = HOLD2||SUBSTR(HOLD1,N+1);
                            CONTENT(I) = HOLD2;
                            M = P + 2;
                            Q = Q + 2;
                            END;
                            ELSE M = M + 1:
                        END:
                ENTER(4): END:
                MAXFORM = I;
                GOTO SEGMENT(12):

/*   PROCESS SCREENING INFORMATION    */

      SEGMENT(5): GET EDIT (CARD) (COLUMN(1),A(80)):
                CALL PACKHOLD:
                J = LENGTH(HOLD) - 1:
                HOLD = SUBSTR(HOLD,1,J):
                COLUMN = 1;
                DO I = 1 TO ENTITY;
                    SEEK(I) = NIL;
                    LOCATOR(I) = BLANK;
                    FALS(I),TRU(I),ENTRY(I) = 0:
                    END;
                SEEKER = NIL;
                M = 0;

/*   SET UP STRUCTURING TABLE    */

                DO WHILE (COLUMN ¬> J);
                    Z = SUBSTR(HOLD,COLUMN,1):
```

```
IF Z = BLANK THEN DO WHILE (Z = BLANK);
          COLUMN = COLUMN + 1;
          Z = SUBSTR(HOLD,COLUMN,1);
          END;
TEMP = NIL;
IF Z = '''' THEN DO;
          C = INDEX(SUBSTR(HOLD,COLUMN+1),'''');
          IF C = 0 THEN DO;
                    PUT SKIP LIST (HOLD);
                    PUT SKIP EDIT
                    ('DELIMITER MISSING IN ABOVE')
                    (COLUMN(1),A);
                    STOP;
                    END;
          TEMP = SUBSTR(HOLD,COLUMN,C+1);
          COLUMN = COLUMN + C + 1;
          GOTO OUT34;
          END;
IF Z < 'A' THEN DO;
          COLUMN = COLUMN + 1;
          TEMP = Z;
          GOTO OUT34;
          END;
DO WHILE (COLUMN ¬> J);
          IF Z < 'A' THEN GOTO OUT34;
          TEMP = TEMP||Z;
          COLUMN = COLUMN + 1;
          IF COLUMN > J THEN GOTO OUT34;
          Z = SUBSTR(HOLD,COLUMN,1);
          END;
OUT34: IF TEMP = ')' | TEMP = '(' THEN
          SEEKER = SEEKER||TEMP;
IF TEMP = 'OR' | TEMP = 'AND' | TEMP = 'NOT'
          THEN SEEKER = SEEKER||SUBSTR(TEMP,1,1);
IF TEMP = 'IN' | TEMP = 'AT' | TEMP = 'GT' |
          TEMP = 'LT' | TEMP = 'GE' | TEMP = 'LE' |
          TEMP = 'EQ' | TEMP = 'NE' THEN DO;
                    LOCATOR(M) = TEMP;
                    N = M - 1;
                    DO WHILE (N>0 & LOCATOR(N)=BLANK);
                              LOCATOR(N)=TEMP;
                              N = N - 1;
                              IF N=0 THEN GOTO ENTER(5);
                              END;
                    END;
IF SUBSTR(TEMP,1,1) = '''' THEN DO;
```

```
                             M = M + 1;
                             SEEKER = SEEKER||SLASH;
                             SEEK(M) = SUBSTR(TEMP,2,LENGTH(TEMP)-2):
                             END;
                        ELSE DO I = 1 TO 8;
                             IF TEMP = SUBSTR(ATTRIBUTES(I),1,
                                  LENGTH(TEMP)) THEN DO:
                                       ENTRY(M) = I:
                                       N = M - 1;
                                       DO WHILE (N>0 & ENTRY(N) =0);
                                            ENTRY(N) = I:
                                            N = N - 1;
                                            IF N=0 THEN GOTO ENTER(5):
                                            END;
                                       GOTO ENTER(5);
                                       END;
                             END;
                        ENTER(5): END:

/*    PARENTHESIZE SCREEN VARIABLE    */

                   HOLD = SEEKER;
                   COLUMN = 1;
                   J = LENGTH(HOLD);
                   DO WHILE (COLUMN ¬> J);
                        C = INDEX(SUBSTR(HOLD,COLUMN),'N');
                        IF C = 0 THEN GOTO OUT3;
                        CALL RIGHTSIDE;
                        SEEKER = SUBSTR(HOLD,1,COLUMN+C-2)||'(';
                        SEEKER = SEEKER||SUBSTR(HOLD,COLUMN+C-1);
                        HOLD = SEEKER;
                        COLUMN = COLUMN + C + 2:
                        J = J + 2:
                        END;
                   OUT3: N = J;
                   DO WHILE (N ¬= 0);
                        IF SUBSTR(HOLD,N,1) = 'N' THEN DO:
                             C = N + 1;
                             IF SUBSTR(HOLD,C,1) = SLASH THEN
                                  SEEKER = SUBSTR(HOLD,1,N-1)||SLASH
                                  ||'N'||SUBSTR(HOLD,C+1);
                                  ELSE DO;
                                  M = 1;
                                  C = C + 1;
                                  DO WHILE (M ¬= 0);
```

```
                    IF SUBSTR(HOLD,C,1) = ')'
                         THEN M = M - 1;
                    IF SUBSTR(HOLD,C,1) = '('
                         THEN M = M + 1;
                    C = C + 1;
                    END;
               SEEKER = SUBSTR(HOLD,1,N-1)||
               SUBSTR(HOLD,N+1,C-N-1);
               SEEKER = SEEKER||'N'|| SUBSTR(HOLD,
                    C);
               END;
          END;
     N = N - 1;
     END;
HOLD = SEEKER;
COLUMN = 1;
DO WHILE (COLUMN ¬> J);
     C = INDEX(SUBSTR(HOLD,COLUMN),'A');
     IF C = 0 THEN GOTO OUT4;
     CALL RIGHTSIDE;
     CALL LEFTSIDE;
     COLUMN = COLUMN + C + 2;
     J = J + 2;
     END;
OUT4: COLUMN = 1;
DO WHILE (COLUMN ¬> J);
     C = INDEX(SUBSTR(HOLD,COLUMN),'O');
     IF C = 0 THEN GOTO OUT18;
     CALL RIGHTSIDE;
     CALL LEFTSIDE;
     COLUMN = COLUMN + C + 2;
     J = J + 2;
     END;
OUT18: SEEKER = '('||SEEKER||')';
M = 0;
N = 0;
COLUMN = 1;
J = LENGTH(SEEKER);
DO WHILE (COLUMN ¬> J);
     Z = SUBSTR(SEEKER,COLUMN,1);
     IF Z = '(' THEN M = M + 1;
     IF Z = ')' THEN M = M - 1;
     IF M = 0 THEN GOTO OUT10;
     IF Z = SLASH THEN DO;
          N = N + 1;
          L = 2;
```

```
                                CALL NEXTLINE;
                                IF L < 0 THEN FALS(N) =L: ELSE FALS(N)=G:
                                L = 1;
                                CALL NEXTLINE:
                                IF L < 0 THEN TRU(N) = L: ELSE TRU(N) =G:
                                END:
                           COLUMN = COLUMN + 1:
                           END:
                 OUT10: CALL GOTCA:
                 GOTO SEGMENT(A):

/*    PROCESS TIME CONTROL CARD    */

  SEGMENT(6): TIMES = TIME:
              HOLD1 = SUBSTR(CARD,15);
              GOTO SEGMENT(11):
    ENTER(6): HH = SUBSTR(TIMES,1,2):
              JJ = SUBSTR(TIMES,3,2):
              KK = SUBSTR(TIMES,5,2):
              LL = SUBSTR(TIMES,7):
              STORAGE = LL + 1000*KK + 60000*JJ + 3600000*HH:
              CLOCK = STORAGE - CLOCK;
              LL = CLOCK / 1000:
              KK = CLOCK - 1000*LL:
              TEMP = LL;
              HOLD2 = KK:
              HOLD2 = SUBSTR(HOLD2,7,3):
              HOLD1 = TEMP||'.'||HOLD2||HOLD1:
              CLOCK = STORAGE:
              PUT SKIP(LINES) EDIT(HOLD1) (COLUMN(COLUMNS),A):
              F = F + LINES:
              IF F > 59 THEN F = F - 59;
              GOTO SEGMENT(12):

/*    PROCESS HEADING CONTROL CARD    */

  SEGMENT(7): GOTO SEGMENT(11):
    ENTER(7): GET EDIT (CARD) (COLUMN(1),A(80)):
              PUT SKIP(LINES) EDIT (CARD) (COLUMN(COLUMNS),A):
              F = F + LINES:
              IF F > 59 THEN F = F - 59;
              GOTO SEGMENT(12):
```

```
/*     PROCESS PRINTCOUNTER CONTROL CARD       */

        SEGMENT(8): HOLD1 = SUBSTR(CARD,13);
                    GOTO SEGMENT(11);
          ENTER(8): NUMBER = COUNTER;
                    NUMBER = SUBSTR(NUMBER,8 - SIZE + 1,SIZE);
                    HOLD1 = NUMBER||HOLD1;
                    PUT SKIP(LINES) EDIT (HOLD1) (COLUMN(COLUMNS),A);
                    F = F + LINES;
                    IF F > 58 THEN F = F - 58;
                    COUNTER = 0;
                    GOTO SEGMENT(12);


/*     PROCESS ENDJOB CONTROL CARD       */

        SEGMENT(9): PUT EDIT ('END OF JOB') (COLUMN(20),A) PAGE;
                    GOTO SEGMENT(12);


/*     PROCESS ORDER CONTROL CARD       */

        SEGMENT(10): C = INDEX(SUBSTR(CARD,7),'BY');
                    COLUMN = 8 + C;
                    DO WHILE (SUBSTR(CARD,COLUMN,1) = BLANK);
                        COLUMN = COLUMN + 1;
                        END;
                    TEMP = NIL;
                    DO WHILE (SUBSTR(CARD,COLUMN,1) ¬= BLANK);
                        Z = SUBSTR(CARD,COLUMN,1);
                        TEMP = TEMP||Z;
                        COLUMN = COLUMN + 1;
                        END;
                    DO K = 1 TO ENTITY;
                        IF TEMP = ATTRIBUTES(K) THEN GOTO OUT1;
                        END;
              OUT1: C = 0;

/*     SET UP ORDER VECTOR    */

                    DO I = 1 TO NO_RCRDS-1;
                        M = ORDER(I);
                        N = ORDER(I+1);
                       LL = TABLE(M,ENTITY2);
                       KK = TABLE(M,K);
                       JJ = TABLE(M,K+1);
                       HOLD3 = SUBSTR(WORD(LL),KK,JJ-KK);
                        IF LENGTH(HOLD3) = 0 THEN HOLD3 = '999999999';
                       LL = TABLE(N,ENTITY2);
                       KK = TABLE(N,K);
                       JJ = TABLE(N,K+1);
                       HOLD4 = SUBSTR(WORD(LL),KK,JJ-KK);
                        IF LENGTH(HOLD4) = 0 THEN HOLD4 = '999999999';
                        IF HOLD3 > HOLD4 THEN DO;
                             C = 1;
                             ORDER(I) = N;
                             ORDER(I+1) = M;
                             END;
                        END;
                   IF C = 0 THEN GOTO SEGMENT(12);
                   GOTO CUT1;

/*     ROUTINE TO GET FORMATTING INFORMATION    */

       SEGMENT(11):PAGES = 1;
                   COLUMNS,LINES = 1;
                   GET DATA (COLUMNS,LINES,PAGES);
                   IF PAGES ¬= 0 THEN DO I = 1 TO PAGES;
                        PUT PAGE;
                        F = 0;
                        END;
                   GOTO ENTER(A);

/*     ROUTINE TO GET A NEW CONTROL CARD AND BRANCH      */

       SEGMENT(12):GET EDIT (CARD) (COLUMN(1),A(80));
                   CALL GOTOA;
                   GOTO SEGMENT(A);

/*     PROCEDURE TO LOOKUP A CONTROL CARD    */

       GOTOA: PROCEDURE;
              DO A = 1 TO 11;
                   IF SUBSTR(CARD,2,7) = CONTROL(A) THEN DO;
                   IF A = 11 THEN GOTO SETUPS; ELSE RETURN; END;
                   END;
              PUT SKIP DATA (CARD);
              PUT SKIP EDIT ('INVALID CONTROL CARD') (COLUMN(20),A);
```

```
            STOP;
 END GOTCA;

/*    PROCEDURE TO PACK DATA INTO HOLD     */

 PACKHOLD: PROCEDURE;
           HOLD,SERIAL = NIL;
           P = 0;
           COLUMN = 1;
           SERIAL = SUBSTR(CARD,IDCOL,MM);
           OUT21: J = IDCOL - 1;
           DO WHILE (SUBSTR(CARD,J,1) = BLANK);
                J = J - 1;
                END;
           HOLD = HOLD||SUBSTR(CARD,COLUMN,J-P);
           IF SUBSTR(CARD,J,1) ¬= SLASH THEN HOLD=HOLD||BLANK;
           GET EDIT (CARD) (COLUMN(1),A(80));
           IF SUBSTR(CARD,1,4) ¬= BLANKS THEN DO;
                HOLD=SUBSTR(HOLD,1,LENGTH(HOLD)-1)||SLASH;
                 J = LENGTH(HOLD);
                 COLUMN = 1;
                 RETURN;
                 END;
           COLUMN = 6;
           P = 5;
           GOTO OUT21;
 END PACKHOLD;

/*    PROCEDURE TO INSERT PARENS ON RIGHT SIDE OF EXPRESSION    */

 RIGHTSIDE: PROCEDURE;
            IF SUBSTR(HOLD,COLUMN+C,1) = SLASH THEN DO;
                 SEEKER = SUBSTR(HOLD,1,COLUMN+C)||')';
                 SEEKER=SEEKER||SUBSTR(HOLD,COLUMN+C+1);
                 END;
                 ELSE DO;
                 N = 1;
                 M = COLUMN + C + 1;
                 DO WHILE (N ¬= 0);
                      IF SUBSTR(HOLD,M,1) = ')' THEN N = N-1;
                      IF SUBSTR(HOLD,M,1) = '(' THEN N = N + 1;
                      M = M + 1;
                      END;
                 SEEKER = SUBSTR(HOLD,1,M-1)||')';
                 SEEKER = SEEKER||SUBSTR(HOLD,M);
                 END;
              HOLD = SEEKER;
              END RIGHTSIDE;

/*   PROCEDURE TO INSERT PARENS ON LEFT SIDE OF EXPRESSION     */

     LEFTSIDE: PROCEDURE;
                  IF SUBSTR(HOLD,COLUMN+C-2,1)=SLASH THEN DO;
                      SEEKER = SUBSTR(HOLD,1,COLUMN+C-3)||'(';
                      SEEKER=SEEKER||SUBSTR(HOLD,COLUMN+C-2);
                      END;
                      ELSE DO;
                      N = 1;
                      M = COLUMN + C - 3;
                      DO WHILE (N¬=0);
                          IF SUBSTR(HOLD,M,1)=')' THEN N=N+1;
                          IF SUBSTR(HOLD,M,1)='(' THEN N=N-1;
                          M = M - 1;
                          END;
                      SEEKER = SUBSTR(HOLD,1,M)||'(';
                      SEEKER = SEEKER||SUBSTR(HOLD,M+1);
                      END;
                  HOLD = SEEKER;
                  END LEFTSIDE;

/*   PROCEDURE TO OVERPRINT A LINE    */

     OPRINT: PROCEDURE;
           IF OVERPRINT(M) = SLASH THEN N=COLOR; ELSE N=1;
           PUT SKIP(TOPLINE(M) + 1);
           DO W = 1 TO N;
               PUT SKIP(0) EDIT (OUTLINE(49)) (COLUMN(1),A);
               END;
           F = F + TOPLINE(M) + 1;
           END OPRINT;

/*   PROCEDURE TO EVALUATE BOOLEAN EXPRESSIONS    */

     NEXTLINE: PROCEDURE;
             C = M;
             G = N;
             H = COLUMN + 1;
```

```
DO WHILE (H ¬> J):
    TEMP = SUBSTR(SEEKER,H,1):
    IF TEMP = ')' THEN DO:
        C = C - 1:
        IF C = 0 THEN DO:
            L = -L:
            RETURN:
            END:
        GOTO OUT16:
        END:
    IF TEMP = '(' THEN DO:
        C = C + 1:
        GOTO OUT16:
        END:
    IF TEMP = SLASH THEN DO:
        G = G + 1:
        RETURN:
        END:
    IF TEMP = 'N' THEN DO:
        IF L = 1 THEN L = 2: ELSE L = 1:
        GOTO OUT16:
        END:
    IF TEMP = 'A' THEN DO:
        IF L = 2 THEN GOTO OUT17:
        GOTO OUT16:
        END:
    IF TEMP = 'C' THEN DO:
        IF L = 1 THEN GOTO OUT17:
        GOTO OUT16:
        END:

OUT17: D = C:
    H = H + 1:
    IF D = 1 THEN DO:
        L = - L:
        RETURN:
        END:
    DO WHILE (H ¬> J):
        TEMP = SUBSTR(SEEKER,H,1):
        IF TEMP = ')' THEN C = C - 1:
        IF TEMP = '(' THEN C = C + 1:
        IF C < D THEN GOTO OUT16:
        IF TEMP = SLASH THEN G = G + 1:
        H = H + 1:
        END:
            CUT1A: H = H + 1;
                    END;
               END NEXTLINE;

     END START;

COMPLETED: END GIPSY;
```

LIST OF REFERENCES

1. Heidorn, George E., *A User's Manual for the Basic Information Processor (BIP)*, 1 June 1966.

2. International Business Machines Corporation Form C28-6571-4, *IBM System/360 Operating System PL/I Language Specifications*, 1965.

3. International Business Machines Corporation Form C28-6594-3, *IBM System/360 Operating System PL/I (F) Programmers Guide*, 1967.

4. International Business Machines Corporation Form C28-8201-1, *IBM System/360 PL/I Reference Manual*, 1968.

5. International Business Machines Corporation, *Generalized Information System Application Description*, 1965.

6. Markowitz, H. M., Hausner, B., and Karr, H. W., *SIMSCRIPT A Simulation Programming Language*, Prentice-Hall, Inc., 1963.

7. McGee, W. C., "Generalized File Processing," *Annual Review in Automatic Programming 5*, p. 77-149, 1969.

8. Northwestern University Report AF-AFSOR-68-1598, *INFOL For the CDC 6400 Information Storage and Retrieval System*, by B. Mittman, et al., 1 November 1968.

9. Olle, T. W., *The Role of Generalized Information Retrieval Systems as an Aid to Decision Making*, paper presented at the International Meeting of the Institute of Management Sciences, 13th, Philadelphia, Pa., 6-8 September 1966.

10. Reitman, W. R., and others, "Autonote: A Personal Information Storage and Retrieval System," *Proceedings of 24th National Conference Association Computing Machinery*, ACM Publication P-69, p. 67-75, 1969.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chief of Naval Operations (OP-91) Department of the Navy Washington, D. C. 20350	1
4.	Asst. Professor G. Heidorn (Code 55 Hd) Department of Operations Analysis Naval Postgraduate School Monterey, California 93940	16
5.	Professor D. G. Williams, Code 0211 Computing Center Naval Postgraduate School Monterey, California 93940	1
6.	LCDR John D. Cooper, USN 1504 West 17th Pine Bluff, Arkansas 71601	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified
	2b. GROUP

3. REPORT TITLE

THE GENERAL INFORMATION PROCESSING SYSTEM (GIPSY)

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*

Master's thesis; December 1969

5. AUTHOR(S) *(First name, middle initial, last name)*

John Duffield Cooper, Jr., Lieutenant Commander, United States Navy

6. REPORT DATE December 1969	7a. TOTAL NO. OF PAGES 108	7b. NO. OF REFS 10
8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO		
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	
d.		

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

This paper describes the General Information Processing System, a computer program designed to serve a variety of information processing applications. An input deck to the program is composed of a data base and a description of the processing tasks to be performed on that data. A typical task would be to screen the data base according to given criteria and then output information from the data that met the criteria. For output, the system has flexible formatting capabilities.

Included in the paper, in Section II, is an example of a bibliographical application, complete with a listing of the input deck and the output that was produced. Sections III through VII contain detailed instructions on how to prepare an input deck. A description of system implementation is contained in Section VIII.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
INFORMATION RETRIEVAL						
GENERAL INFORMATION PROCESSING						
NON-ARITHMETIC PROCESSING						

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

THE GENERAL INFORMATION PROCESSING SYSTEM (GIPSY)

4. DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*

Master's thesis; December 1969

5. AUTHOR(S) *(First name, middle initial, last name)*

John Duffield Cooper, Jr., Lieutenant Commander, United States Navy

6. REPORT DATE December 1969	7a. TOTAL NO. OF PAGES 108	7b. NO. OF REFS 10

8a. CONTRACT OR GRANT NO. b. PROJECT NO c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

13. ABSTRACT

This paper describes the General Information Processing System, a computer program designed to serve a variety of information processing applications. An input deck to the program is composed of a data base and a description of the processing tasks to be performed on that data. A typical task would be to screen the data base according to given criteria and then output information from the data that met the criteria. For output, the system has flexible formatting capabilities.

Included in the paper, in Section II, is an example of a bibliographical application, complete with a listing of the input deck and the output that was produced. Sections III through VII contain detailed instructions on how to prepare an input deck. A description of system implementation is contained in Section VIII.

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0101-807-6811

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
INFORMATION RETRIEVAL						
GENERAL INFORMATION PROCESSING						
NON-ARITHMETIC PROCESSING						

DD FORM 1 NOV 65 1473 (BACK)
S/N 0101-807-6821

thesC75404
The general information processing syste
3 2768 002 09410 4
DUDLEY KNOX LIBRARY